

Applicant(s): Dietz, *et al.*

Application No.: 10/684,776

Filed: October 14, 2003

Title: METHOD AND APPARATUS FOR
MONITORING TRAFFIC IN A NETWORK

Group Art Unit: 2157

Examiner: Moustafa M. Meky

DECLARATION UNDER 37 CFR 1.131

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

1. I am an inventor of claims 11–59 of the above referenced patent application.
2. Claims 11–59 have been rejected under 35 USC 102(e) as anticipated by Muller et al. (U.S. Patent 6,483,804) that has a reference date of March 1, 1999.
3. Prior to the reference date of March 1, 1999, I conceived of the invention of claims 11, 29, and 54 shown in the following:
 - **Exhibit A0:** Directory of documents
 - **Exhibit A1:** Technically Elite MeterFlow Accelerator Modules System Specification (Document MFASystem.pdf)
 - **Exhibit A2:** Technically Elite MeterFlow Accelerator Parser Module Specification (Document MFAParser.pdf)
 - **Exhibit A3:** Technically Elite MeterFlow Accelerator Analyzer Module Specification (Document MFAAnalyze.pdf)
 - **Exhibit A4:** Protocol Tracking Summary (Document MFAProtocolLayout.pdf)

A copy of each of Exhibits A0 to A4 is attached. The dates on each such copy has been deleted. **I confirm that the dates are all prior to March 1, 1999.** These exhibits are as follows.

Exhibit A0 is a dated computer directory of documents that describe the design and tests to run the design on real data.

Exhibit A1 is the overall design of the system that implements the method claims 11 and 54, and includes the elements of claim 29.

Exhibit A2 is a detailed design of the parsing/extraction unit that carries out step (b) of claim 11, that corresponds to element (c): the parser subsystem of claim 29, and that carries out the parsing/extraction operations of element (b) of claim 54.

Exhibit A3 is a detailed design of the analyzer that carries out the operations of elements (c), (d), and (e) of method claim 29, that corresponds to elements (d), (e) and (f) unit parsing/extraction unit that carries out carries out the operations of elements (c), (d), and (e) of method claim 54, that corresponds to element (c): the parser subsystem of claim 29, and that carries out the parsing/extraction operations of element (b) of claim 54.

Exhibit A4 is a summary of the protocols that the system can analyze.

Note that Technically Elite was the name of the predecessor of the assignee of the present invention at the time.

3. Prior to the reference date of March 1, 1999, I reduced to practice the invention of claims 11, 29, and 54 of the above referenced patent application as shown in the following documents:

- **Exhibit B0** is a dated computer directory of test data and documents used therefore.
- **Exhibit B1:** Technically Elite MeterFlow Accelerator Modules Testbench Specification (Document MFATest.pdf in directory of Exhibit A0)
- **Exhibit B2:** The first page of file big.cpl.

The cpl files (big.cpl, bigfgc3.cpl, bigfgpc.cpl, bigfpayl.cpl, bigfpayl2.cpl, bigfpgrp.cpl, bigfpgrp2.cpl, bigfrag.cpl, bigfrag2.cpl, output.cpl, Protocols.cpl, short.cpl, shrtfpg2.cpl, shrtfps3.cpl, shrtfps4.cpl, shrtfps5.cpl, shrttunl.cpl) are files for the protocol compiler of all the actual protocols recognized by the system. These files include a description of the parser information for the parser to perform the parsing/extracting operation according to the protocol. They also contain the state processing states for the state operations of elements (d) and (e) of claim 54. The first page of one file is provided.

- **Exhibit B3:** The first four pages of a printout of file MFATEST.HEX that contains the actual packets captured by the packet acquisition device described in element (a) of claims 11 and 54, and corresponding to the contents of element (b), the input buffer memory of claim 29. The packet acquisition device for the experiment was a SUN workstation connected to a connection point of a network.
- **Exhibit B4:** The file packets.txt that describes the nature of the packets in MFATEST.HEX.
- **Exhibit B5:** The contents of files mfaptpkt.txt and mfaptpkt2.txt that are files that contain the elements that were extracted by the parsing/extracting of
- **Exhibit B6:** The contents of files mfaptkey.txt and mfaptkey2.txt that are files that contain the keys that were generated from the extracted data (Exhibit B4) and used for looking up the flow-entry database per element (c) of method claims 11 and 54, which are operations carried out by the lookup engine of element (e) of claim 29.

- **Exhibit B7:** The first four pages of a printout of file MFATEST.TXT that includes the decoded packets that were generated by operation of the method that includes the elements of each of method claims 11 and 54, by an apparatus that includes the elements of claim 29.
- **Exhibit B8:** Protocol Definition Language (PDL) Reference Guide (the document MFS-PDL-Reference.pdf) that provides a reference to the protocol definition language used in cpl files.
- **Exhibit B9:** State-based Sub-Classification Overview (document MFS-State-Classification.pdf) that describes the states of some of the protocols that are supported.

The invention functioned for its intended purpose by running the apparatus on a computer, and a program implementing the method on test data that was part of a node of a network.

The above exhibits are each a copy. The date on each copy has been deleted. I confirm that the deleted dates are each prior to March 1, 1999.

Therefore, and in summary, I declare that the inventions of claims 11, 29, and 54 were reduced to practice prior to the reference data of March 1, 1999.

I hereby declare under penalty of perjury under the laws of the United States of America that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Signed,



March 1, 2005

Date

Russell S. Dietz

Address for correspondence:

Dov Rosenfeld
5507 College Avenue, Suite 2,
Oakland, CA 94618
Tel. 510-547-3378
Fax: +1-510-291-2985; Email: dov@inventek.com

- **Exhibit A0:**Directory of documents

Hardware Specification-dir.txt
Directory of M:\aaa-----INVENTEK CLIENTS\Hifn\Patents\APPT-001-1-1 filed
Proof of Reductn to Practice\Hardware Specification\

M:\aaa-----INVENTEK CLIENTS\Hifn\Patents\APPT-001-1-1 filed Proof of
Reductn to Practice\Hardware Specification\

```
=====
=====
MFAAnalyze.pdf          317 KB      04:47:46 AM      a
MFAApp1.pdf             12 KB      05:50:46 AM      a
MFAParser.pdf           124 KB      03:56:18 AM      a
MFAProtocolLayout.pdf   13 KB      09:24:38 AM      a
MFASystem.pdf            93 KB      03:56:50 AM      a
MFATest.pdf             18 KB      03:56:26 AM      a
```

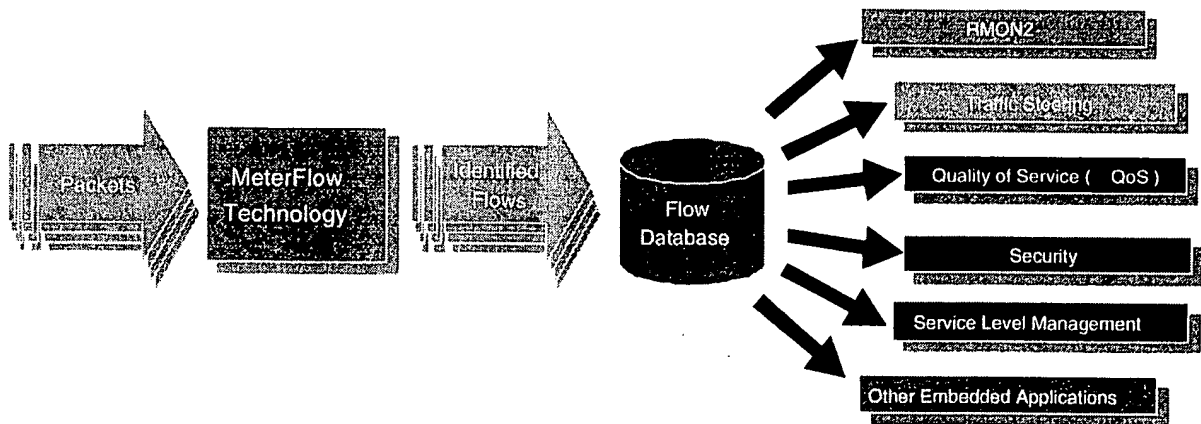
AA
Total 0 folder(s); 6 file(s)

Total files size: 1 MB; 580 KB; 593910 Bytes

AA

- **Exhibit A1:** Technically Elite MeterFlow Accelerator Modules System Specification (Document MFASystem.pdf)

Technically Elite MeterFlow Accelerator Modules System Specification



1 Introduction

The Technically Elite MeterFlow Accelerator is a set of synthesizable modules designed to do wire speed hardware based application traffic recognition for Fast Ethernet and Gigabit Ethernet. Originally designed for RMON2 network management the MeterFlow Accelerator also allows Layer 3 (Network) through Layer 7 (Application) visibility for switches and routers. The MeterFlow Accelerator poaches the network traffic and builds a “flow” database that is then extracted for further processing. Each flow consists of the information necessary to track the conversation between the two end points of the traffic. This conversation is also characterized and vital statistics counted. The resulting flow database is useful for many applications. Some of these include RMON2 network management, traffic steering, quality of service, security, and service level management.

1.1 Technically Elite MeterFlow Accelerator Highlights

- Synthesizable modules written in both the Verilog and VHDL
- Processes up to Gigabit speeds
- Complete traffic data
- State based parallel processing architecture
- Distributes work to eliminate bottlenecks
- Layer 3 network protocols to dynamic transaction oriented applications at Layer 7
- Scalable architecture for any size switch or probe
- Can recognize over 2000 different protocols
- Extensible to new protocols
- Recognizes encapsulations
- Open interface
- Easy to use software tools including protocol compiler and C model

2 Overview

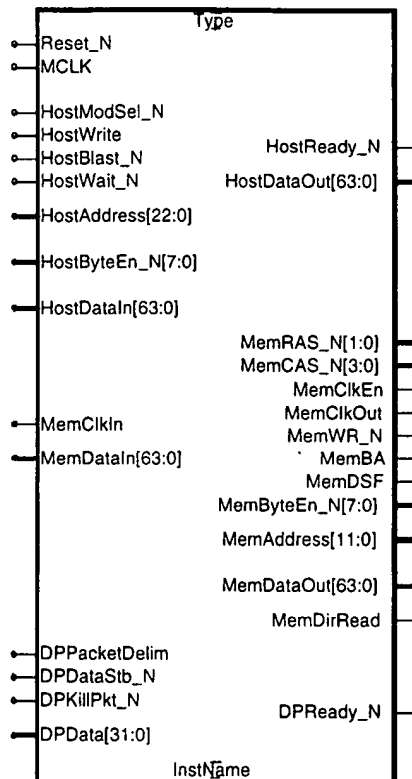
The Technically Elite MeterFlow Accelerator Modules System Specification outlines the general system requirements. It provides an overview of how the modules interact with each other and external devices. It also provides guidance for the testing methodology to be used in the verification of the cores.

The Technically Elite network analysis suite consists of three main components. These are the parser, the analyzer and software. The parser works on the information contained in a single packet. The analyzer builds flow information across multiple packets. The software consists of a compiler, a C model and a database of protocol information. The database delineates all the information needed by the parser to recognize the protocols and build the flow key. The database also delineates how each protocol's flow entry should be updated as well as the procedure to recognize multi-packet protocols (state processing). Also included in the module set is a host interface module. This module defines a burst oriented bus interface compatible with the Intel i960. This module can be easily modified to interface to other bus types.

After initialization the network data first goes to the parser. The parser attempts to recognize the various possible protocols in a particular packet. It then builds a flow key data structure that is passed to the analyzer. The analyzer first attempts to find a particular packets related flow in its' database. Then using the information it gathered from previous packets in this flow and the current packets' data it updates the flows' data base entry. Once a flow has been completely recognized, updates consist of gathering statistics. On a regular basis the external system reads the flow data base for further processing.

The parser and analyzer modules are RTL synthesizable modules written in both the Verilog and VHDL hardware description languages. Each major component of the cores has a matching testbench. The testbenches fully exercise the unit under test and provide an automated verification environment. Input stimulus files are automatically generated by the compiler and expected data files are automatically generated by the C model.

3 Top Level MeterFlow Accelerator Module Symbol



4 Top Level MeterFlow Accelerator Module Pin Descriptions

4.1.1.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low. When this signal is active the module sets it's registers to their default condition and suspends operation. It will only respond to host access cycles. The DataPort interface will keep DPReady_N active to avoid problems for the external circuitry.
MCLK	IN	1	Module Clock. All internal and external transfers except for memory transfers are synchronized by this signal.

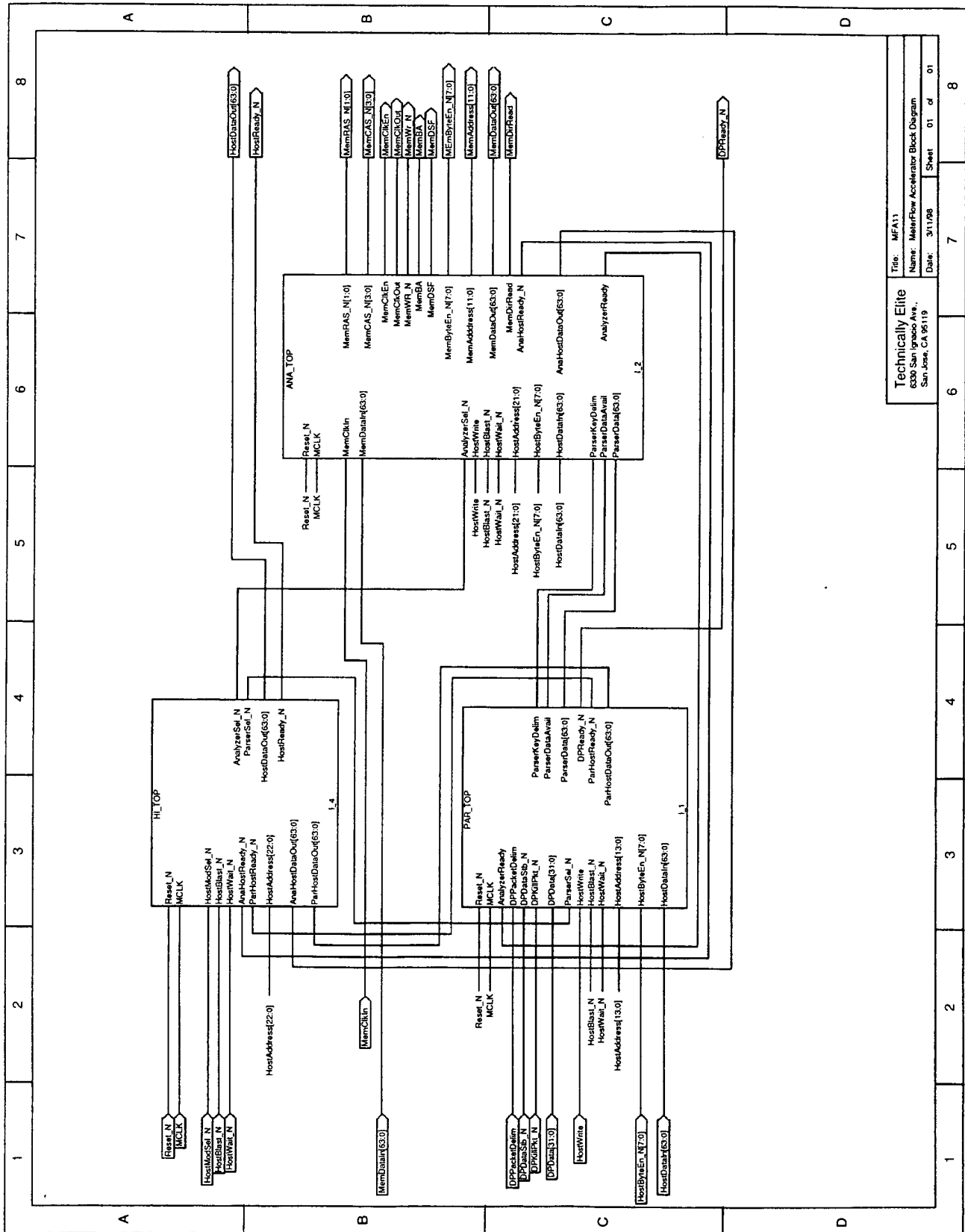
4.1.1.2 Memory Interface			
Signal	Dir	Width	Description
MemClkIn	IN	1	Memory clock in. This signal is used to generate the memory interface timing.
MemRAS_N	OUT	2	Memory Row Address Strobe bus – active low.
MemCAS_N	OUT	4	Memory Column Address Strobe bus– active low.
MemClkEn	OUT	1	Memory Clock Enable. Some memories require this signal to be disabled for a certain amount of time after reset.
MemClkOut	OUT	1	Memory Clock Out. This signal is used by synchronous memory for all operations. MemClkIn is buffered and sent out on this pin. This helps reduce skew between this clock and the other signals.
MemWR_N	OUT	1	Memory Write – active low.
MemBA	OUT	1	Memory Bank Address. Used by multi-bank memory to select the bank the current operation is to operate on.
MemDSF	OUT	1	Memory Special Function select.
MemByteEn_N	OUT	8	Memory Byte Enable bus– active low.
MemAddress	OUT	12	Memory Address bus.
MemDataIn	IN	64	Memory Data Input bus.
MemDataOut	OUT	64	Memory Data Output bus.
MemDirRead	OUT	1	Memory Data bus Direction is Read. This signal is used to control the tri-state enable on the bidirectional memory data bus. If MemDirRead is active data is coming into the module from the memory. If it is inactive the module is driving data out to the memory.

4.1.1.3 Host Interface Signals			
Signal	Dir	Width	Description
HostModSel_N	IN	1	Host interface Module Select - active low. HostModSel_N is sampled on the rising edge of MCLK . If it is active, it signifies that the external host is attempting to access the module.
HostWrite	IN	1	Write. Write is sampled on the rising edge of MCLK . This signal is only valid when HostModSel_N is active. If this signal is active, the host is attempting to write to the module. Inactive this signal signifies a read from the module. It should also be used to control the direction of the host data bus if it is bidirectional.
HostBlast_N	IN	1	Burst Last – active low. HostBlast_N is sampled on the rising edge of MCLK . HostBlast_N tells the module that the current transfer is the last transfer in this burst.
HostWait_N	IN	1	Wait – active low. HostWait_N is sampled on the rising edge of MCLK . The host asserts HostWait_N when it wishes to slow transfers between itself and the module. This could also be used by additional interface logic to slow transfers so it can multiplex the bus down to a smaller size without additional FIFOs. If wait is active, HostReady_N is blocked.
HostReady_N	OUT	1	Ready – active low. HostReady_N should be sampled on the rising edge of MCLK . The module returns HostReady_N when the current cycle is completed. For a write operation, HostReady_N means that the HostDataIn bus has been latched. For a read operation HostReady_N means that the requested data is on the HostDataOut bus and is valid. HostReady_N is blocked by HostWait_N .
HostAddress	IN	23	Host Address bus. HostAddress is sampled on the rising edge of MCLK if HostModSel_N is active. This bus defines the first address in this burst to access in the 64 Megabyte address space of the module. See Section x.x.x for the Address Utilization Map.
HostByteEn_N	IN	8	Host Byte Enable bus – Active low. HostWait_N is sampled on the rising edge of MCLK .
HostDataIn	IN	64	Host Data Input bus. HostDataIn is sampled on the rising edge of MCLK if HostWrite is active and HostWait_N is inactive.
HostDataOut	OUT	64	Host Data Output bus. HostDataOut should be sampled on the rising edge of MCLK . Data on this bus is valid during a read cycle when HostReady_N is active.

4.1.1.4 Data Port Interface			
Signal	Dir	Width	Description
DPPacketDelim	IN	1	Data Port Packet Delimiter. This signal should be driven active when the external logic wants to send a packet to the module. DPPacketDelim should remain active during the entire packet transfer. DPPacketDelim must go inactive for one clock between packets.
DPDataStb_N	IN	1	Data Port Data Strobe. When active, this signal tells the module that data on the DPData bus is valid. If DPReady_N was inactive at the end of the previous cycle, DPDataStb_N should not be driven active. If DPReady_N goes inactive in the same cycle as DPDataStb_N , then the module will latch the incoming data so that no data is lost.
DPKillPkt_N	IN	1	Data Port Kill Packet. If this signal becomes active while DPPacketDelim is active, the module will attempt to stop processing the current packet and flush it's input FIFO. If however, parsing of the packet is completed, the packet will not be able to be recalled. This should only be a problem in a 'cut through' implementation.
DPReady_N	OUT	1	Data Port Ready – active low. This signal when driven active means that the module can accept new data. If however the modules' input FIFO is filled, DPReady_N will be driven inactive. To prevent overruns, DPReady_N will go inactive when the module can actually accept one more data transfer.
DPData	IN	32	Data Port Data bus.

5 MeterFlow Accelerator Modules Block Diagrams

The following page is the top level block diagram for the MeterFlow Accelerator Module.



Title: MFA11		Sheet	01	of	01
Name: MeterFlow Accelerator Block Diagram					
Date: 3/1/98					

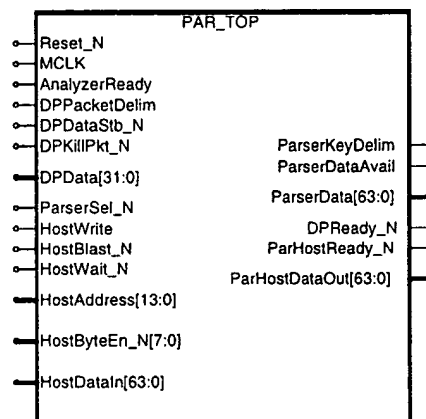
6 Description of Modules and Software

6.1 Parser Module

6.1.1 Parser Module Highlights

- Builds key and payload data structure for analyzer (flow key)
- Scaleable protocol pattern recognition engine
- Supports from 1 to 2048 simultaneous unique protocol patterns
- At 62.5 MegaHertz can process up to 1.5 MegaPackets per second
- Accepts protocol database output from MeterFlow compiler

6.1.2 Parser Module Symbol



6.1.3 Parser Module Description

The parser module consists of two main sub-modules. These are the pattern recognition engine and the slicer. The parser module pushes the network data through the DataPort interface. The data is first processed by the pattern recognition engine. This engine consists of a database and a comparison engine. The database can reside in ROM or RAM. If the database is in a RAM the parser can be programmed to recognize new protocols or a different set of protocols.

The set of specified protocols defines a tree of linked nodes. Each protocol is either a parent node or a terminal node. A protocol is a parent node if it links to other protocols that can be contained in it. For example IP is a parent to UDP. As each protocol is recognized, the pattern recognition engine emits a unique protocol identifier. It also emits a process code that the slicer uses to build the flow key.

The slicer extracts information from the packet to build the flow key. For example, it will extract the source and destination addresses from the packet and pack them into the flow key data structure. It may also process certain parts of the packet to speed up flow processing performed by the analyzer. It will build a hash value from certain parts of the packet to speed looking up the flow in the analyzers' database.

6.1.4 Parser Module Pin Descriptions

6.1.4.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low. When this signal is active the parser sets it's registers to their default condition and suspends operation. It will only respond to host access cycles. The DataPort interface will keep DPReady_N active to avoid problems for the external circuitry.
MCLK	IN	1	Module Clock. All internal and external transfers except for memory transfers are synchronized by this signal.

6.1.4.2 Analyzer Interface			
Signal	Dir	Width	Description
AnalyzerReady	IN	1	Analyzer Ready. This signal tells the parser that the analyzer can accept data.
ParserKeyDelim	OUT	1	Parser Key Delimiter. The ParserKeyDelim signal becomes active when the first quadword of a new key is ready to transfer to the analyzer. It goes inactive when the last quadword of the key is transferred.
ParserDataAvail	OUT	1	Parser Data Available. If this signal is active the data on the ParserData bus is valid.
ParserData	OUT	64	Parser Data bus.



6.1.4.3 Data Port Interface			
Signal	Dir	Width	Description
DPPacketDelim	IN	1	Data Port Packet Delimiter. This signal should be driven active when the external logic wants to send a packet to the parser. DPPacketDelim should remain active during the entire packet transfer. DPPacketDelim must go inactive for one clock between packets.
DPDataStb_N	IN	1	Data Port Data Strobe. When active, this signal tells the parser that data on the DPData bus is valid. If DPReady_N was inactive at the end of the previous cycle, DPDataStb_N should not be driven active. If DPReady_N goes inactive in the same cycle as DPDataStb_N , then the parser will latch the incoming data so that no data is lost.
DPKillPkt_N	IN	1	Data Port Kill Packet. If this signal becomes active while DPPacketDelim is active, the parser will attempt to stop processing the current packet and flush it's input FIFO. If however, parsing of the packet is completed, the packet will not be able to be recalled. This should only be a problem in a 'cut through' implementation.
DPReady_N	OUT	1	Data Port Ready – active low. This signal when driven active means that the parser can accept new data. If however the parser's input FIFO is filled, DPReady_N will be driven inactive. To prevent overruns, DPReady_N will go inactive when the parser can actually accept one more data transfer.
DPData	IN	32	Data Port Data bus.

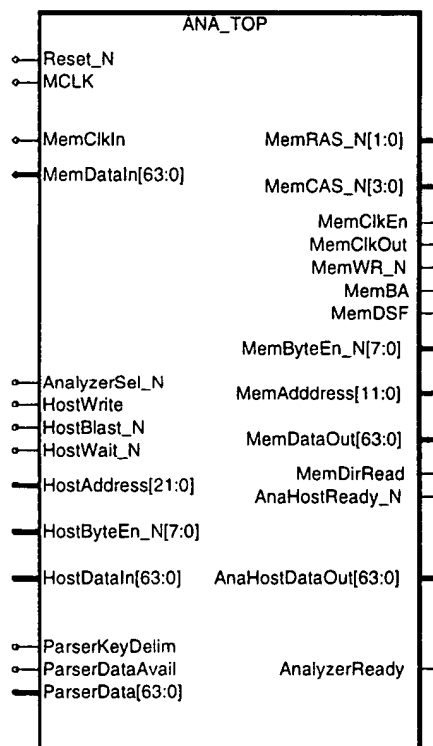
6.1.4.4 Host Interface Signals			
Signal	Dir	Width	Description
ParserSel_N	IN	1	Parser Select - active low. ParserSel_N is sampled on the rising edge of MCLK . If it is active, it signifies that the external host is attempting to access the parser.
HostWrite	IN	1	Write. Write is sampled on the rising edge of MCLK . This signal is only valid when ParserSel_N is active. If this signal is active, the host is attempting to write to the parser. Inactive this signal signifies a read from the parser.
HostBlast_N	IN	1	Burst Last – active low. HostBlast_N is sampled on the rising edge of MCLK . HostBlast_N tells the parser that the current transfer is the last transfer in this burst.
HostWait_N	IN	1	Wait – active low. HostWait_N is sampled on the rising edge of MCLK . The host asserts HostWait_N when it wishes to slow transfers between itself and the parser.
ParHostReady_N	OUT	1	Parser to Host Ready – active low. ParHostReady_N should be sampled on the rising edge of MCLK . The parser returns ParHostReady_N when the current cycle is completed. For a write operation, ParHostReady_N means that the HostDataIn bus has been latched. For a read operation ParHostReady_N means that the requested data is on the ParHostDataOut bus and is valid. ParHostReady_N is blocked by HostWait_N .
HostAddress	IN	13	Host Address bus. HostAddress is sampled on the rising edge of MCLK if ParserSel_N is active. This bus defines the first address in this burst to access in the 64 Kilobyte address space of the Parser. See Section x.x.x for the Address Utilization Map.
HostByteEn_N	IN	8	Host Byte Enable bus – Active low. HostWait_N is sampled on the rising edge of MCLK .
HostDataIn	IN	64	Host Data Input bus. HostDataIn is sampled on the rising edge of MCLK if HostWrite is active and HostWait_N is inactive.
ParHostDataOut	OUT	64	ParserHost Data Output bus. ParHostDataOut should be sampled on the rising edge of MCLK . Data on this bus is valid during a read cycle when ParHostReady_N is active.

6.2 Analyzer Module

6.2.1 Analyzer Module Highlights

- “Flexible” Rule-based Traffic Classification
- State-based Tracking of Traffic
- **Multiple** Packets for Layer Processing
- Internal Cache and Memory Controller (32 - 64KB)
- Direct High Bandwidth (64 bit) Memory Interface
- Up to 16MB of memory (75K Flows)
- SG/SDRAM Support
- Programmable Rules/State Engine
- Selectable Protocols in Flows
- Future Protocols Support
- Scalable System Design

6.2.2 Analyzer Module Symbol



6.2.3 Analyzer Module Description

The analyzer module consists of the flow lookup engine, the flow insertion/deletion engine, the simple rules engine, the complex rules engine, the caching memory controller, the host update controller and the process synchronizer. Each of these sub-modules work in parallel to create and update flows.

As a flow key enters the analyzer, the lookup engine attempts to find it in the flow database. If the flow exists, the lookup engine retrieves the flow from the caching memory controller. It then makes a decision based on the state information included in the flow entry to either send it to the simple rules engine, the complex rules engine or to update the flow entry itself. This updating consists of adding values to counters in the flow database entry. If a flow does not exist, the flow key is sent to the flow insertion/deletion engine which adds the flow to the database. Based on the flow key information the flow insertion/deletion engine may be also send the new flow to one of the rules engines for processing.

The simple rules engine updates the flow based on the current state and the flow key information. The complex rules engine processes multi packet protocol recognition. It may have to search through a series of possible states to determine the flow's actual state. The result of the complex engine's processing is a consolidated flow entry. For example, a PointCast session will open multiple conversations that on a packet by packet basis look like separate flows. Since each conversation is merely a subflow under the PointCast master flow, a single flow that consolidates all of the information for the flow is desired.

The caching memory controller can be setup to work with various configurations of SDRAM or SGRAM. It uses it's cache to optimize memory bandwidth. On a typical network the packets will have a certain amount of congruity. This means that the cache can have a high hit rate.

6.2.4 Analyzer Module Pin Out

6.2.4.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low. When this signal is active the analyzer sets it's registers to their default condition and suspends operation. It will only respond to host access cycles.
MCLK	IN	1	Module Clock. All internal and external transfers except for memory transfers are synchronized by this signal.

6.2.4.2 Memory Interface			
Signal	Dir	Width	Description
MemClkIn	IN	1	Memory clock in. This signal is used to generate the memory interface timing.
MemRAS_N	OUT	2	Memory Row Address Strobe bus – active low.
MemCAS_N	OUT	4	Memory Column Address Strobe bus– active low.
MemClkEn	OUT	1	Memory Clock Enable. Some memories require this signal to be disabled for a certain amount of time after reset.
MemClkOut	OUT	1	Memory Clock Out. This signal is used by synchronous memory for all operations. MemClkIn is buffered and sent out on this pin. This helps reduce skew between this clock and the other signals.
MemWR_N	OUT	1	Memory Write – active low.
MemBA	OUT	1	Memory Bank Address. Used by multi-bank memory to select the bank the current operation is to operate on.
MemDSF	OUT	1	Memory Special Function select.
MemByteEn_N	OUT	8	Memory Byte Enable bus– active low.
MemAddress	OUT	12	Memory Address bus.
MemDataIn	IN	64	Memory Data Input bus.
MemDataOut	OUT	64	Memory Data Output bus.
MemDirRead	OUT	1	Memory Data bus Direction is Read. This signal is used to control the tri-state enable on the bidirectional memory data bus. If MemDirRead is active data is coming into the analyzer from the memory. If it is inactive the analyzer is driving data out to the memory.

6.2.4.3 Host Interface Signals			
Signal	Dir	Width	Description
AnalyzerSel_N	IN	1	Host interface Analyzer Select - active low. AnalyzerSel_N is sampled on the rising edge of MCLK . If it is active, it signifies that the external host is attempting to access the analyzer.
HostWrite	IN	1	Write. Write is sampled on the rising edge of MCLK . This signal is only valid when AnalyzerSel_N is active. If this signal is active, the host is attempting to write to the analyzer. Inactive this signal sign signifies a read from the analyzer. It should also be used to control the direction of the host data bus if it is bidirectional.
HostBlast_N	IN	1	Burst Last – active low. HostBlast_N is sampled on the rising edge of MCLK . HostBlast_N tells the analyzer that the current transfer is the last transfer in this burst.
HostWait_N	IN	1	Wait – active low. HostWait_N is sampled on the rising edge of MCLK . The host asserts HostWait_N when it wishes to slow transfers between itself and the analyzer. This could also be used by additional interface logic to slow transfers so it can multiplex the bus down to a smaller size without additional FIFOs. If wait is active, HostReady_N is blocked.
AnaHostReady_N	OUT	1	Analyzer to Host Ready – active low. AnaHostReady_N should be sampled on the rising edge of MCLK . The analyzer returns AnaHostReady_N when the current cycle is completed. For a write operation, AnaHostReady_N means that the HostDataIn bus has been latched. For a read operation AnaHostReady_N means that the requested data is on the HostDataOut bus and is valid. AnaHostReady_N is blocked by HostWait_N .
HostAddress	IN	22	Host Address bus. HostAddress is sampled on the rising edge of MCLK if AnalyzerSel_N is active. This bus defines the first address in this burst to access in the 32 Megabyte address space of the analyzer. See Section x.x.x for the Address Utilization Map.
HostByteEn_N	IN	8	Host Byte Enable bus – Active low. HostWait_N is sampled on the rising edge of MCLK .
HostDataIn	IN	64	Host Data Input bus. HostDataIn is sampled on the rising edge of MCLK if HostWrite is active and HostWait_N is inactive.
AnaHostDataOut	OUT	64	Analyzer Host Data Output bus. AnaHostDataOut should be sampled on the rising edge of MCLK . Data on this bus is valid during a read cycle when AnaHostReady_N is active.

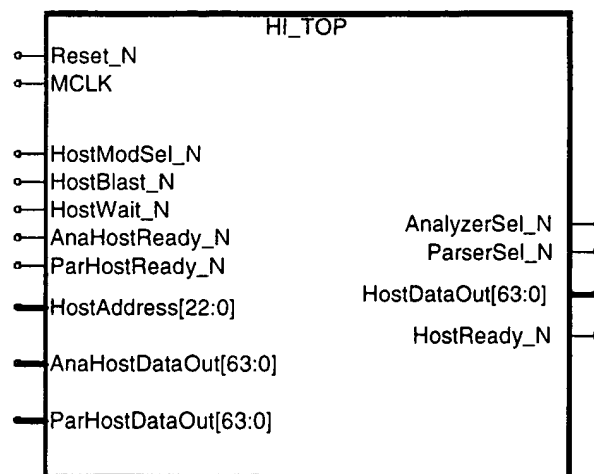
6.2.4.4 Parser Interface			
Signal	Dir	Width	Description
AnalyzerReady	OUT	1	Analyzer Ready. This signal tells the parser that the analyzer can accept data.
ParserKeyDelim	IN	1	Parser Key Delimiter. The ParserKeyDelim signal becomes active when the first quadword of a new key is ready to transfer to the analyzer. It goes inactive when the last quadword of the key is transferred.
ParserDataAvail	IN	1	Parser Data Available. If this signal is active the data on the ParserData bus is valid.
ParserData	IN	64	Parser Data bus.

6.3 Host Interface Module

6.3.1 Host Interface Module Highlights

- i960 style burst interface
- Easily modified for connection to other buses

6.3.2 Host Interface Symbol



6.3.3 Host Interface Module Description

The Host Interface module contains the host data multiplexer to select either the parser or the analyzer data bus. It also decodes the host address to create **ParserSel_N** or **AnalyzerSel_N**.

6.3.4 Host Interface Module Pin Out

6.3.4.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low. When this signal is active the analyzer sets it's registers to their default condition and suspends operation. It will only respond to host access cycles.
MCLK	IN	1	Module Clock. All internal and external transfers except for memory transfers are synchronized by this signal.

6.3.4.2 Host Interface Signals			
Signal	Dir	Width	Description
AnalyzerSel_N	OUT	1	Host interface Analyzer Select - active low. AnalyzerSel_N is sampled on the rising edge of MCLK . If it is active, it signifies that the external host is attempting to access the analyzer.
ParserSel_N	OUT	1	Parser Select - active low. ParserSel_N is sampled on the rising edge of MCLK . If it is active, it signifies that the external host is attempting to access the parser.
HostBlast_N	IN	1	Burst Last – active low. HostBlast_N is sampled on the rising edge of MCLK . HostBlast_N tells the analyzer that the current transfer is the last transfer in this burst.
HostWait_N	IN	1	Wait – active low. HostWait_N is sampled on the rising edge of MCLK . The host asserts HostWait_N when it wishes to slow transfers between itself and the analyzer. This could also be used by additional interface logic to slow transfers so it can multiplex the bus down to a smaller size without additional FIFOs. If wait is active, HostReady_N is blocked.
AnaHostReady_N	IN	1	Analyzer to Host Ready – active low. AnaHostReady_N should be sampled on the rising edge of MCLK . The analyzer returns AnaHostReady_N when the current cycle is completed. For a write operation, AnaHostReady_N means that the HostDataIn bus has been latched. For a read operation AnaHostReady_N means that the requested data is on the HostDataOut bus and is valid. AnaHostReady_N is blocked by HostWait_N .
ParHostReady_N	IN	1	Parser to Host Ready – active low. ParHostReady_N should be sampled on the rising edge of MCLK . The parser returns ParHostReady_N when the current cycle is completed. For a write operation, ParHostReady_N means that the HostDataIn bus has been latched. For a read operation ParHostReady_N means that the requested data is on the ParHostDataOut bus and is valid. ParHostReady_N is blocked by HostWait_N .
HostReady_N	OUT	1	Ready – active low. HostReady_N should be sampled on the rising edge of MCLK . The module returns HostReady_N when the current cycle is completed. For a write operation, HostReady_N means that the HostDataIn bus has been latched. For a read operation HostReady_N means that the requested data is on the HostDataOut bus and is valid. HostReady_N is blocked by HostWait_N .
HostAddress	IN	23	Host Address bus. HostAddress is sampled on the rising edge of MCLK if AnalyzerSel_N is active. This bus defines the first address in this burst to access in the 64 Megabyte address space of the analyzer. See Section x.x.x for the Address Utilization Map.

AnaHostDataOut	IN	64	Analyzer Host Data Output bus. AnaHostDataOut should be sampled on the rising edge of MCLK. Data on this bus is valid during a read cycle when AnaHostReady_N is active.
ParHostDataOut	IN	64	ParserHost Data Output bus. ParHostDataOut should be sampled on the rising edge of MCLK. Data on this bus is valid during a read cycle when ParHostReady_N is active.
HostDataOut	OUT	64	Host Data Output bus. HostDataOut should be sampled on the rising edge of MCLK. Data on this bus is valid during a read cycle when HostReady_N is active.

6.4 MeterFlow Compiler

6.4.1 MeterFlow Compiler Highlights

- ANSI compatible C implementation
- Simple Packet Description Language
- Technically Elite supplied Packet Description Language files for all common protocols
- Any or all protocols can be included
- Automatically generates parser module pattern recognition database
- Automatically generates slicer instructions
- Automatically generates unique protocol identifiers for use throughout system
- Automatically generates analyzer programming
- Automatically generates test input stimulus

6.4.2 MeterFlow Compiler Description

The MeterFlow Compiler generates all the information needed to program the MeterFlow accelerator. It's input is a set of files that define the protocols to recognize and the target system. It can also be used by the engineer to define the size of the databases required to support a certain set of protocols. The output of the compiler is a set of files used to program each part of the MeterFlow Accelerator.

The compiler first reads the protocol definition files defined in the protocol list file and creates a tree defining each protocols relationship to the others. Protocols with the same name are assumed to be the same. For example, FTP under UDP and TCP are condensed into a single entry linked to both parent protocols. The compiler then reads the hardware definition file or uses a default maximum definition. It then searches the protocol space to find a solution. If a solution is found that fits into the hardware constraints, the compiler outputs database in a form that can be loaded into either the testbenches, the C model, or the hardware.

If the t option is selected, the compiler will generate an input stimulus file for the testbenches. This file contains a series of packets one for each protocol in the protocol list.

6.4.3 MeterFlow Compiler Invocation

MFC <options>

6.4.3.1 Options

Option	Name	Description
i < filename>	Protocol list filename	The protocol list file contains the names of each protocol to be included in this run. The default is MeterFlow.pl. The names must match the filename prefix of the protocol definition language file associated with that protocol. For example, if the TCP protocol is to be included, and the file is called TCP.PDL, the protocol list file should contain the line: I TCP; If the children of TCP are to be included they can be added automatically by replacing the above line with: I TCP c; Child protocols can be excluded by the following line as a example: E FTP;

o <prefix>	Output file prefix	The output file prefix allows the user to change the start of the filename of the output files. The default is MeterFlow.
d <filename>	Hardware definition filename	This input file is used by the compiler to constrain processing to the available hardware resources. If the compiler cannot find a solution at the effort level it will output a set of files with the best solution it found and report an error.
e <n>	Effort	N is a number from 1 to 5. The default is 3. An effort level of 5 tells the compiler to exhaust the search space.
t	Generate input stimulus file	This option generates a file that can be run through the C model to generate expected output data. Then both files can be run through the testbenches for automated testing of the modules.

6.5 MeterFlow C Model

6.5.1 MeterFlow C Model Highlights

- ANSI compatible C implementation
- Models the MeterFlow Accelerator modules
- Outputs expected data for the testbenches
- Expects the same input files as the testbenches

6.5.2 MeterFlow C Model Description

The MeterFlow C Model reads the same files used by the modules and emulates them. It is used to generate expected data for the automated testbenches included with the modules.

6.5.3 MeterFlow C Model Invocation

MCM <options>

6.5.3.1 Options

Option	Name	Description
i < filename>	Input filename prefix	The input file prefix allows the user to change the start of the filename of the input files. The default is MeterFlow.
o <prefix>	Output file prefix	The output file prefix allows the user to change the start of the filename of the output files. The default is MeterFlow.
d <filename>	Hardware definition filename	This input file is used by the C model to emulate the available hardware resources.

7 MeterFlow Accelerator Single Chip Implementation Top Level Schematic



- **Exhibit A2:** Technically Elite MeterFlow Accelerator Parser Module Specification
(Document MFAParser.pdf)

Technically Elite MeterFlow Accelerator Parser Module Specification

Not For External Release!

Revision History		
Version	Date	Description
0.9	[REDACTED]	First release
1.0	[REDACTED]	Rev 1

0 Table of Contents

0	Table of Contents	2
1	Introduction	5
2	Technically Elite MeterFlow Accelerator Parser Module Highlights	5
3	Architectural Overview	6
3.1	Bandwidth requirements	7
3.2	Architectural Block Diagram	8
4	Top Level MeterFlow Accelerator Parser Module Symbol	9
5	MeterFlow Accelerator Parser Module Top Level Pin Descriptions	10
5.1.1.1	General Interface Signals	10
5.1.1.2	Analyzer Interface	10
5.1.1.3	DataPort Interface	11
5.1.1.4	Host Interface Signals	12
6	MeterFlow Accelerator Parser Module Top Level VHDL Entity	13
7	MeterFlow Accelerator Parser Module Top Level Verilog Module	13
8	MeterFlow Accelerator Parser Module Top Level Schematic	16
9	Parser Module Constants Files	17
9.1	Parser module Verilog Constants File – ParserConstants.v	17
9.2	Parser module VHDL Constants File – ParserConstants.vhd	17
10	Sub-module Descriptions	18
10.1	Pattern Recognition Engine Sub-module – PRE	18
10.1.1	Symbol	18
10.1.2	Highlights	18
10.1.3	Description	18
10.1.4	Search Algorithm Psuedo-code	18
10.1.5	Implementation Information	18
10.1.5.1	Database Word Definition	18
10.1.6	File Names	18
10.1.7	Pin Descriptions	19
10.1.7.1	General Interface Signals	19
10.1.7.2	Slicer Interface	19
10.1.7.3	CPU Interface MUX Interface	19
10.1.7.4	Parser Input Buffer Interface	19
10.1.8	Verilog Module	19
10.2	Slicer Sub-module	21
10.2.1	Symbol	21
10.2.2	Description	21
10.2.2.1	Instruction Word Definition	21
10.2.3	Implementation Information	21
10.2.4	File Names	21
10.2.5	Pin Descriptions	22
10.2.5.1	General Interface Signals	22
10.2.5.2	Parser Input Buffer Interface	22
10.2.5.3	Parser Output Buffer Interface	22
10.2.5.4	CPU Interface MUX Interface	22
10.2.5.5	Pattern Recognition Engine Interface	22
10.2.5.6	Analyzer Interface Control Interface	24
10.2.6	Verilog Module	24

10.3	Pattern Recognition Database Sub-module - PRD	25
10.3.1	Symbol	25
10.3.2	Highlights	25
10.3.3	Description	25
10.3.4	Implementation Information	25
10.3.5	File Names	25
10.3.6	Pin Descriptions	25
10.3.6.1	CPU Interface MUX Interface	25
10.3.7	Verilog Module	25
10.4	Slicer Instruction Database Sub-module -SID	26
10.4.1	Symbol	26
10.4.2	Highlights	26
10.4.3	Description	26
10.4.4	Implementation Information	26
10.4.5	File Names	26
10.4.6	Pin Descriptions	26
10.4.6.1	CPU Interface MUX Interface	26
10.4.7	Verilog Module	26
10.5	CPU Interface MUX and Control Register Sub-module - CMC	28
10.5.1	Symbol	28
10.5.2	Description	28
10.5.3	File Names	28
10.5.4	Pin Descriptions	28
10.5.4.1	General Interface Signals	28
10.5.4.2	Slicer Instruction Database Interface	28
10.5.4.3	Pattern Recognition Database Interface	28
10.5.4.4	Slicer Interface	29
10.5.4.5	Pattern Recognition Engine Interface	29
10.5.4.6	Host Interface Signals	29
10.5.5	Verilog Module	30
10.6	Parser Input Buffer Sub-module – PIB	32
10.6.1	Symbol	32
10.6.2	Highlights	32
10.6.3	Description	32
10.6.4	Implementation Information	32
10.6.5	File Names	33
10.6.6	Pin Descriptions	33
10.6.6.1	General Interface Signals	33
10.6.6.2	DataPort Interface	33
10.6.6.3	DataPort Interface Control Interface	33
10.6.6.4	Pattern Recognition Engine Interface	33
10.6.6.5	Slicer Interface	34
10.6.7	Verilog Module	34
10.7	Parser Output Buffer Sub-module - POB	36
10.7.1	Symbol	36
10.7.2	Highlights	36
10.7.3	Description	36
10.7.4	Implementation Information	36
10.7.5	File Names	36
10.7.6	Pin Descriptions	37
10.7.6.1	General Interface Signals	37
10.7.6.2	Slicer Interface	37
10.7.6.3	Analyzer Interface Control Interface	37
10.7.6.4	Analyzer Interface	37

10.7.7	Verilog Module.....	38
10.8	DataPort Interface Control Sub-module - DPIC	39
10.8.1	Symbol	39
10.8.2	Description	39
10.8.3	Implementation Information	39
10.8.4	File Names	39
10.8.5	Pin Descriptions	39
10.8.5.1	General Interface Signals	39
10.8.5.2	DataPort Interface	39
10.8.5.3	Parser Input Buffer Interface	40
10.8.5.4	Pattern Recognition Engine Interface	40
10.8.6	Verilog Module.....	40
10.9	Analyzer Interface Control Sub-module -AIC.....	42
10.9.1	Symbol	42
10.9.2	Description	42
10.9.3	Implementation Information	42
10.9.4	File Names	42
10.9.5	Pin Descriptions	42
10.9.5.1	General Interface Signals	42
10.9.5.2	Analyzer Interface.....	42
10.9.5.3	Slicer Interface.....	43
10.9.5.4	Parser Output Buffer Interface.....	43
10.9.6	Verilog Module.....	43



1 Introduction

This document is designed to be the repository for all information related to the MeterFlow Accelerator Parser Module. This specification is designed to provide the engineer with enough information to fully implement the module. There will be revisions during and after the implementation process that will be reflected in this document.

Each part of this specification describes a different aspect of the module. It concentrates on the interfaces between the parser module and the other parts of the system. The other parts of the system include the analyzer module, the host interface module and importantly the software that models, programs and tests the system. The key to a successful implementation is the interfaces between modules and between sub-module and sub-module. Each interface is described in detail. Any changes to the interfaces may affect the entire module and even the entire system. Care must be taken that each interface is understood completely before implementation is begun.

2 Technically Elite MeterFlow Accelerator Parser Module Highlights

- Synthesizable modules written in both the Verilog and VHDL
- Scalable architecture for any size switch or probe
- Can recognize over 2000 different protocols
- Extensible to new protocols
- Recognizes encapsulations
- Builds key and payload data structure for analyzer (flow key)
- Scaleable protocol pattern recognition engine
- At 62.5 MegaHertz can process up to 1.5 MegaPackets per second
- Accepts protocol database output from MeterFlow compiler

3 Architectural Overview

The parser module consist of two main sub-modules. These are the pattern recognition engine (PRE) and the slicer. The PRE analyzes the packet and the slicer builds the flow key from the packet and instructions from the pattern recognition engine. The parser has been split into two parts for several reasons. First and foremost, the split correctly partitions the functions to allow maximum reuse of silicon across the over two thousand protocols that can be supported. Another advantage of the split architecture is that the compiler can analyze the three dimensional space occupied by the offset, level, and pattern data of the specified protocols and compact the databases used in the parser module. The set of specified protocols defines a tree of linked nodes. Each protocol is either a parent node or a terminal node. A protocol is a parent node if it links to other protocols that can be contained in it. For example IP is a parent to UDP. Protocols can be the children of several parents. If a unique node was generated for each of the possible parent/child trees, the database would explode exponentially. Instead, child nodes are shared among multiple parents thus compacting the database.. Finally the PRE can be used on it's own when only protocol recognition is required.

The parser module pouches the network data through the DataPort interface. The data is first processed by the pattern recognition engine. This engine consists of a comparison engine and a database. The comparison engine has a first stage that checks the protocol type field to determine if it is an 802.3 packet and the field should be treated as a length. If it is not a length, the protocol is checked in the second stage. This is the only protocol level that is not programmable. This is because the detection of the protocol at this level is simple and well defined. It is implemented with partial CAMs that return a node identifier if hit. This second stage has two full sixteen bit CAMs defined for future protocol additions. After this detection is completed the engine initializes Current Offset Pointer (COP) to the next part of the packet that needs to be checked. The node identifier from the previous stage and the data pointed to by the COP are used by the PRE to lookup an entry in the database. As each protocol is recognized, the pattern recognition engine emits a unique protocol identifier. It also emits a process code that the slicer uses to build the flow key. This process is repeated until the node identifier's Terminal bit is set. At that point the PRE has completely recognized the protocols in the packet and readies itself for the next packet.

The slicer extracts information from the packet to build the flow key. For example, it will extract the source and destination addresses from the packet and pack them into the flow key data structure. It may also process certain parts of the packet to speed up flow processing performed by the analyzer. It will build a hash value from certain parts of the packet to speed looking up the flow in the analyzers' database. The slicer transfers data from it's input Buffer to it's output Buffer based on the sequence of instructions in it's instruction database. When the PRE recognizes a protocol it outputs both the protocol identifier and a process code to the slicer. The protocol identifier is added to the flow key and the process code is used to fetch the first instruction from the instruction database. Instructions consist an operation code and usually source and destination offsets as well as a length. The offsets and length are in bytes. A typical operation is the MOVE instruction. This instruction tells the slicer to copy n bytes data unmodified from the input Buffer to the output Buffer. The slicer contains a byte-wise barrel shifter so that the bytes moved can be packed into the flow key. The slicer contains another instruction called HASH. This instruction tells the slicer to copy from the input Buffer to the HASH generator. The result from the HASH generator is always written into the first two bytes of the flow key. It is used to accelerate the lookup of the flow in the analyzers flow database. Once the flow key is completed, the slicer transfers it to the analyzer for further processing.

The parser module databases can reside in ROM or RAM. If the databases are in a RAM the parser can be programmed to recognize new protocols or a different set of protocols.

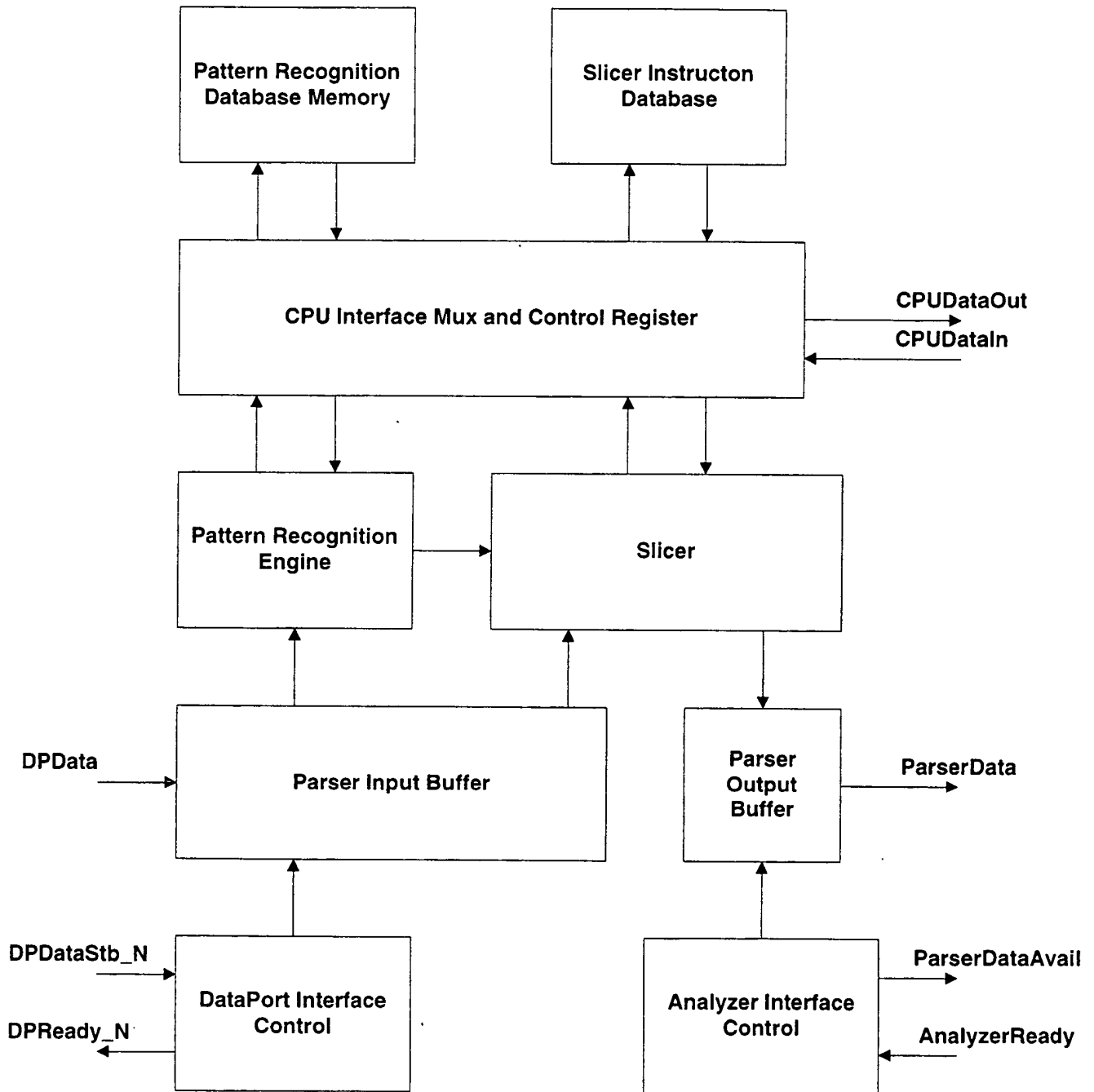
3.1 Bandwidth requirements

The target throughput for the MeterFlow Accelerator running at 62.5 Megahertz is 1.5 million packets per second (PPS). This is the sustained maximum throughput of a single Gigabit channel. At this rate the parser module has 41.6 cycles to process each packet. In order to reduce the need for front end buffering external to the parser module, the architecture has been designed to complete the protocol recognition generation in no more than 36 cycles. Since there could be up to 12 different protocols in each to be processed, the parser module has been designed to average three cycles per protocol. This is the very worst case because a packet that has twelve levels of protocols in it will most likely be much larger than the minimum packet size. This can be used as to advantage again in the reduction of external buffering. The slicer must also complete the flow key generation within 36 cycles to keep the system in balance and unstalled. This however can be extended if the payload copying instructions run to there maximum values.

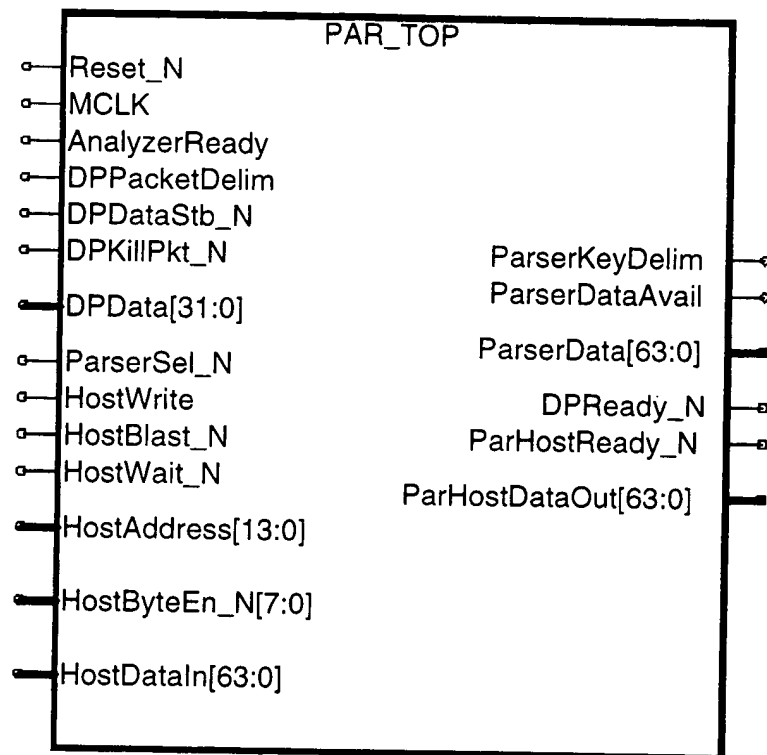
The average packet will have between 4 and 5 levels of protocol with no encapsulations. At three cycles per protocol the PRE will use only 15 cycles to complete a packet. This means that the PRE has a typical sustained throughput of over three million packets per second.



3.2 Architectural Block Diagram



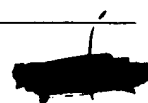
4 Top Level MeterFlow Accelerator Parser Module Symbol



5 MeterFlow Accelerator Parser Module Top Level Pin Descriptions

5.1.1.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low. When this signal is active the parser sets it's registers to their default condition and suspends operation. It will only respond to host access cycles. The DataPort interface will keep DPRReady_N active to avoid problems for the external circuitry.
MCLK	IN	1	Module Clock. All internal and external transfers except for memory transfers are synchronized by this signal.

5.1.1.2 Analyzer Interface			
Signal	Dir	Width	Description
AnalyzerReady	IN	1	Analyzer Ready. This signal tells the parser that the analyzer can accept data.
ParserKeyDelim	OUT	1	Parser Key Delimiter. The ParserKeyDelim signal becomes active when the first quadword of a new key is ready to transfer to the analyzer. It goes inactive when the last quadword of the key is transferred.
ParserDataAvail	OUT	1	Parser Data Available. If this signal is active the data on the ParserData bus is valid.
ParserData	OUT	64	Parser Data bus.



5.1.1.3 DataPort Interface			
Signal	Dir	Width	Description
DPPacketDelim	IN	1	DataPort Packet Delimiter. This signal should be driven active when the external logic wants to send a packet to the parser. DPPacketDelim should remain active during the entire packet transfer. DPPacketDelim must go inactive for one clock between packets.
DPDataStb_N	IN	1	DataPort Data Strobe. When active, this signal tells the parser that data on the DPData bus is valid. If DPReady_N was inactive at the end of the previous cycle, DPDataStb_N should not be driven active. If DPReady_N goes inactive in the same cycle as DPDataStb_N , then the parser will latch the incoming data so that no data is lost.
DPKillPkt_N	IN	1	DataPort Kill Packet. If this signal becomes active while DPPacketDelim is active, the parser will attempt to stop processing the current packet and flush it's input Buffer. If however, parsing of the packet is completed, the packet will not be able to be recalled. This should only be a problem in a 'cut through' implementation.
DPReady_N	OUT	1	DataPort Ready – active low. This signal when driven active means that the parser can accept new data. If however the parser's input Buffer is filled, DPReady_N will be driven inactive. To prevent overruns, DPReady_N will go inactive when the parser can actually accept one more data transfer.
DPData	IN	32	DataPort Data bus.

5.1.1.4 Host Interface Signals			
Signal	Dir	Width	Description
ParserSel_N	IN	1	Parser Select - active low. ParserSel_N is sampled on the rising edge of MCLK . If it is active, it signifies that the external host is attempting to access the parser.
HostWrite	IN	1	Write. Write is sampled on the rising edge of MCLK . This signal is only valid when ParserSel_N is active. If this signal is active, the host is attempting to write to the parser. Inactive this signal sign signifies a read from the parser.
HostBlast_N	IN	1	Burst Last – active low. HostBlast_N is sampled on the rising edge of MCLK . HostBlast_N tells the parser that the current transfer is the last transfer in this burst.
HostWait_N	IN	1	Wait – active low. HostWait_N is sampled on the rising edge of MCLK . The host asserts HostWait_N when it wishes to slow transfers between itself and the parser.
ParHostReady_N	OUT	1	Parser to Host Ready – active low. ParHostReady_N should be sampled on the rising edge of MCLK . The parser returns ParHostReady_N when the current cycle is completed. For a write operation, ParHostReady_N means that the HostDataIn bus has been latched. For a read operation ParHostReady_N means that the requested data is on the ParHostDataOut bus and is valid. ParHostReady_N is blocked by HostWait_N .
HostAddress	IN	13	Host Address bus. HostAddress is sampled on the rising edge of MCLK if ParserSel_N is active. This bus defines the first address in this burst to access in the 64 Kilobyte address space of the Parser. See Section x.x.x for the Address Utilization Map.
HostByteEn_N	IN	8	Host Byte Enable bus – Active low. HostWait_N is sampled on the rising edge of MCLK .
HostDataIn	IN	64	Host Data Input bus. HostDataIn is sampled on the rising edge of MCLK if HostWrite is active and HostWait_N is inactive.
ParHostDataOut	OUT	64	ParserHost Data Output bus. ParHostDataOut should be sampled on the rising edge of MCLK . Data on this bus is valid during a read cycle when ParHostReady_N is active.

6 MeterFlow Accelerator Parser Module Top Level VHDL Entity

```

entity PAR_TOP is
    Port
    (
        AnalyzerReady : In  std_logic;
        DPDataStb_N : In  std_logic;
        DPData : In  std_logic_vector (31 downto 0);
        DPKillPkt_N : In  std_logic;
        DPPacketDelim : In  std_logic;
        HostAddress : In  std_logic_vector (13 downto 0);
        HostBlast_N : In  std_logic;
        HostByteEn_N : In  std_logic_vector (7 downto 0);
        HostDataIn : In  std_logic_vector (63 downto 0);
        HostWait_N : In  std_logic;
        HostWrite : In  std_logic;
        MCLK : In  std_logic;
        ParserSel_N : In  std_logic;
        Reset_N : In  std_logic;
        DPReady_N : Out  std_logic;
        ParHostDataOut : Out  std_logic_vector (63 downto 0);
        ParHostReady_N : Out  std_logic;
        ParserDataAvail : Out  std_logic;
        ParserData : Out  std_logic_vector (63 downto 0);
        ParserKeyDelim : Out  std_logic
    );
end PAR_TOP;

```

7 MeterFlow Accelerator Parser Module Top Level Verilog Module

```

module par_top( AnalyzerReady, DPData, DPDataStb, DPKillPkt_N, DPPacketDelim, DPReady_N,
    HostAddress, HostBlast_N, HostDataIn, HostWait_N, HostWrite,
    MCLK, ParHostDataOut, ParHostReady_N, ParserData,
    ParserDataAvail, ParserKeyDelim, ParserSel_N, Reset_N );
    input AnalyzerReady;
    input [63:0] DPData;
    input DPDataStb, DPKillPkt_N, DPPacketDelim;
    output DPReady_N;
    input [12:0] HostAddress;
    input HostBlast_N;
    input [63:0] HostDataIn;
    input HostWait_N, HostWrite, MCLK;
    output [63:0] ParHostDataOut;
    output ParHostReady_N;
    output [63:0] ParserData;
    output ParserDataAvail, ParserKeyDelim;
    input ParserSel_N, Reset_N;
    wire [8:0] DPICAdd;
    wire [63:0] PIBuSIData;

```

```

wire [8:0] SIPBAdd;
wire [63:0] PIBuPREData;
wire [8:0] PREnPIBAdd;
wire [29:0] CMCoSIData;
wire [8:0] SIAdd;
wire [3:0] PREnSIProtocol;
wire [63:0] SIPOBData;
wire [5:0] PREnSICommand;
wire [8:0] SIPOBAdd;
wire [8:0] CMCoPRDAdd;
wire [22:0] CMCoPRDDData;
wire [3:0] BaseOffset;
wire [22:0] CMCoPREData;
wire [8:0] PREAdd;
wire [22:0] PRDDData;
wire [29:0] SIData;
wire [8:0] CMCoSIDAdd;
wire [8:0] AICPOBAdd;
wire [29:0] SIDData;
wire [29:0] CMCoSIDData;
wire [8:0] AICoPOBAdd;
wire [8:0] SIFlowKeySize;
wire [8:0] SIPIBAdd;
wire AICDone;
wire CMCoSIDWr;
wire CMCoPRDWr;
wire PREnSIEn;
wire SIWrStb;
wire SIDone;
wire PREDone;
wire DPICWrStb;
wire DPICDone;
wire ParserEn;

```

```

AIC I11 ( .AICDone(AICDone), .AICoPOBAdd(AICoPOBAdd[8:0]),
.AnalyzerReady(AnalyzerReady), .MCLK(MCLK),
.ParserDataAvail(ParserDataAvail), .ParserEn(ParserEn),
.ParserKeyDelim(ParserKeyDelim), .Reset_N(Reset_N), .SIDone(SIDone),
.SIFlowKeySize(SIFlowKeySize[8:0]) );
PRE I10 ( .BaseOffset(BaseOffset[3:0]), .CMCoPREData(CMCoPREData[22:0]),
.MCLK(MCLK), .ParserEn(ParserEn), .PIBuPREData(PIBuPREData[63:0]),
.PREAdd(PREAdd[8:0]), .PREDone(PREDone), .PREnPIBAdd(PREnPIBAdd[8:0]),
.PREnSICommand(PREnSICommand[5:0]), .PREnSIEn(PREnSIEn),
.PREnSIProtocol(PREnSIProtocol[3:0]), .Reset_N(Reset_N) );
DPIC I1 ( .DPDataStb(DPDataStb), .DPICAdd(DPICAdd[8:0]), .DPICDone(DPICDone),
.DPICWrStb(DPICWrStb), .DPKillPkt_N(DPKillPkt_N),
.DPPacketDelim(DPPacketDelim), .DPReady_N(DPReady_N), .MCLK(MCLK),
.ParserEn(ParserEn), .PREDone(PREDone), .Reset_N(Reset_N) );
Slicer I2 ( .CMCoSIData(CMCoSIData[29:0]), .MCLK(MCLK), .ParserEn(ParserEn),
.PIBuSIData(PIBuSIData[63:0]), .PREDone(PREDone),
.PREnSICommand(PREnSICommand[5:0]), .PREnSIEn(PREnSIEn),
.PREnSIProtocol(PREnSIProtocol[3:0]), .Reset_N(Reset_N),
.SIAdd(SIAdd[8:0]), .SIDone(SIDone),
.SIFlowKeySize(SIFlowKeySize[8:0]), .SIPIBAdd(SIPIBAdd[8:0]),

```

```

        .SIPOBAdd(SIPOBAdd[8:0]), .SIPOBData(SIPOBData[63:0]),
        .SIWrStb(SIWrStb) );
SID I3 ( .CMCoSIDAdd(CMCoSIDAdd[8:0]), .CMCoSIDData(CMCoSIDData[29:0]),
        .CMCoSIDWr(CMCoSIDWr), .SIDData(SIDData[29:0]) );
PRD I4 ( .CMCoPRDAdd(CMCoPRDAdd[8:0]), .CMCoPRDDData(CMCoPRDDData[22:0]),
        .CMCoPRDWr(CMCoPRDWr), .PRDDData(PRDDData[22:0]) );
POB I5 ( .AICDone(AICDone), .AICoPOBAdd(AICoPOBAdd[8:0]), .MCLK(MCLK),
        .ParserData(ParserData[63:0]), .ParserEn(ParserEn), .Reset_N(Reset_N),
        .SIDone(SIDone), .SIPOBAdd(SIPOBAdd[8:0]), .SIPOBData(SIPOBData[63:0]),
        .SIWrStb(SIWrStb) );
PIB I6 ( .DPData(DPData[63:0]), .DPICAdd(DPICAdd[8:0]), .DPICDone(DPICDone),
        .DPICWrStb(DPICWrStb), .MCLK(MCLK), .ParserEn(ParserEn),
        .PIBuPREData(PIBuPREData[63:0]), .PIBuSIDData(PIBuSIDData[63:0]),
        .PREDone(PREDone), .PREnPIBAdd(PREnPIBAdd[8:0]), .Reset_N(Reset_N),
        .SIDone(SIDone), .SIPIBAdd(SIPIBAdd[8:0]) );
CMC I8 ( .BaseOffset(BaseOffset[3:0]), .CMCoPRDAdd(CMCoPRDAdd[8:0]),
        .CMCoPRDDData(CMCoPRDDData[22:0]), .CMCoPRDWr(CMCoPRDWr),
        .CMCoPREData(CMCoPREData[22:0]), .CMCoSIDAdd(CMCoSIDAdd[8:0]),
        .CMCoSIDData(CMCoSIDData[8:0]), .CMCoSIDWr(CMCoSIDWr),
        .CMCoSIDData(CMCoSIDData[29:0]), .HostAddress(HostAddress[12:0]),
        .HostBlast_N(HostBlast_N), .HostDataIn(HostDataIn[63:0]),
        .HostWait_N(HostWait_N), .HostWrite(HostWrite), .MCLK(MCLK),
        .ParHostDataOut(ParHostDataOut[63:0]),
        .ParHostReady_N(ParHostReady_N), .ParserEn(ParserEn),
        .ParserSel_N(ParserSel_N), .PRDDData(PRDDData[22:0]),
        .PREAdd(PREAdd[8:0]), .Reset_N(Reset_N), .SIDData(SIDData[29:0]),
        .SIAdd(SIAdd[8:0]) );

```

```
endmodule // par_top
```

8 MeterFlow Accelerator Parser Module Top Level Schematic

Insert Schematic Here



9 Parser Module Constants Files

The parser module constants files contain a list of constants used to allow rapid configuration of the module. For example the size of the slicers instruction database data bus is defined as :

Verilog

```
'define PAR_SLI_DWIDTH 23 // Parser Slicer Instruction Database Data Bus Width
```

VHDL

```
constant PAR_SLI_DWIDTH : integer := 23; -- Parser Slicer Instruction Database Data Bus Width
```

9.1 Parser module Verilog Constants File – ParserConstants.v

```
'define PAR_COM_SHIFT 3 // Parser Command Shift
'define PAR_SLI_DWIDTH 23
'define PAR_DP_DWIDTH 32 // Parser Data Port Data Bus Width
'define PAR_PIB_DWIDTH 64 // Parser Input Buffer Data Bus Width
'define PAR_PIB_AWIDTH 9 // Parser Input Buffer Address Bus Width
'define PAR_PRD_AWIDTH 9 // Parser Pattern Recognition Database Address Bus Width
'define PAR_PRD_DWIDTH 23 // Parser Pattern Recognition Database Data Bus Width
'define PAR_SID_AWIDTH 9 // Parser Slicer Instruction Database Address Bus Width
'define PAR_SID_DWIDTH 30 // Parser Slicer Instruction Database Data Bus Width
'define PAR_POB_DWIDTH 64 // Parser Output Buffer Data Bus Width
'define PAR_POB_AWIDTH 9 // Parser Output Buffer Address Bus Width
'define PAR_BASE_OFF_WIDTH 4 // Parser Base Offset Width
'define PAR_HOST_AWIDTH 13 // Parser Host Address Bus Width
'define PAR_HOST_BE_WIDTH 8 // Parser Host Byte Enable Bus Width
'define PAR_HOST_DWIDTH 64 // Parser Host Data Bus Width
'define PAR_PRE_COM_WIDTH 6 // Parser Command Width
'define PAR_COM_CT_WIDTH 4 // Parser Command Count Width
'define PAR_PRE_PRO_WIDTH 4 // Parser
'define PAR_CONTROL_REG_SIZE 5 // Parser Control Register Size
'define PAR_H_SIDDELTA 34
'define PAR_H_PRDELTA 41
'define PAR_H_CRDELTA 59 // CAN'T BE NESTED!
'define PAR_SL_FKS_WIDTH 9 // Parser Slicer Flow Key Size Width
```

9.2 Parser module VHDL Constants File – ParserConstants.vhd

Insert ParserConstants.vhd here

10 Sub-module Descriptions

10.1 Pattern Recognition Engine Sub-module – PRE

10.1.1 Symbol

10.1.2 Highlights

- Scaleable protocol pattern recognition engine
- Supports from 1 to 2048 simultaneous unique protocol patterns
- At 62.5 MegaHertz can process up to 1.5 MegaPackets per second
- Accepts protocol database output from MeterFlow compiler

10.1.3 Description

The Pattern Recognition Engine module searches it's database and the packet in order to recognize the protocols the packet contains. The database consists of a series of linked lookup tables. Each lookup table uses eight bits of addressing. The first lookup table is always at address zero. The Pattern Recognition Engine uses the **BaseOffset** from the control register to start the comparison. It loads this value into the Current Offset Pointer (COP). It then reads the byte at **BaseOffset** from the Parser Input Buffer and uses it as an address into the first lookup table.

Each lookup table returns a word that links to another lookup table or it returns a terminal flag. If the lookup produces a recognition event the database also returns a command for the Slicer. Finally it returns the value to add to the COP.

10.1.4 Search Algorithm Psuedo-code

10.1.5 Implementation Information

10.1.5.1 Database Word Definition	
Bit	Description
1:0	Opcode 00 Terminal Node found 01 Intermediate Node 10 Ending Terminal Node found
*	Next Lookup table * uses PAR_PRE_LU_WIDTH
*	Slicer Command * uses PAR_PRE_COM_WIDTH
*	Mask * uses PAR_PRE_MASK_WIDTH

10.1.6 File Names

Top: PRE.v(hd)

Uses: ParserConstants.v(hd)

10.1.7 Pin Descriptions

10.1.7.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
PREDone	OUT	1	Pattern Recognition Engine Done.
ParserEn	IN	1	Parser Enable bit from control register

10.1.7.2 Slicer Interface			
Signal	Dir	Width	Description
PREnSIEn	OUT	1	Pattern Recognition Engine to Slicer Enable
PREnSICommand	OUT	*	Pattern Recognition Engine to Slicer Command bus * uses PAR_PRE_COM_WIDTH
PREnSIProtocol	OUT	*	Pattern Recognition Engine to Slicer Protocol bus * uses PAR_PRE_PRO_WIDTH

10.1.7.3 CPU Interface MUX Interface			
Signal	Dir	Width	Description
PREAdd	OUT	*	Pattern Recognition Engine Address bus * uses PAR_PRD_AWIDTH
BaseOffset	IN	4	Base Offset. This is the first offset the Pattern Recognition Engine will check.
CMCoPREData	IN	*	CMC to Pattern Recognition Engine Data bus * uses PAR_PRD_DWIDTH

10.1.7.4 Parser Input Buffer Interface			
Signal	Dir	Width	Description
PREnPIBAdd	OUT	*	Pattern Recognition Engine Parser Input Buffer Address bus. * Uses PAR_PIB_AWIDTH
PIBuPREData	IN	*	Parser Input Buffer to Pattern Recognition Engine Data bus. * Uses PAR_PIB_DWIDTH

10.1.8 Verilog Module

```
/*
PRE.v
PRE - Pattern Recognition Module
```


*/

'include "ParserConstants.v"

module PRE(Reset_N, MCLK, ParserEn, PREDone, PREnSIEn, PREnSICommand, PREnSIProtocol,
PREAdd, BaseOffset, CMCoPREData, PREnPIBAdd, PIBuPREData);

// General Interface Interface

input Reset_N;

input MCLK;

input ParserEn;

output PREDone;

// Slicer Interface

output PREnSIEn;

output ['PAR_PRE_COM_WIDTH-1 : 0] PREnSICommand;

output ['PAR_PRE_PRO_WIDTH-1 : 0] PREnSIProtocol;

// CMC Interface

output ['PAR_PRD_AWIDTH-1 : 0] PREAdd;

input ['PAR_BASE_OFF_WIDTH-1 : 0] BaseOffset;

input ['PAR_PRD_DWIDTH-1 : 0] CMCoPREData;

// Parser Input Buffer Interface

output ['PAR_PIB_AWIDTH-1 : 0] PREnPIBAdd;

input ['PAR_PIB_DWIDTH-1 : 0] PIBuPREData;



10.2 Slicer Sub-module

10.2.1 Symbol

10.2.2 Description

The Slicer cuts up the packet to build the flow key. The Slicer module accepts commands from the Pattern Recognition Engine. Based on the command received, the Slicer either transfers data from the Parser Input Buffer to the Parser Output Buffer or it transfers data from the Parser Input Buffer to it's internal hash generator. It contains a buffer that FIFO's up the commands. When the Pattern Recognition Engine asserts **PREDone** the Slicer completes any pending commands, transfers the hash to the Parser Output Buffer and asserts **SIDone**.

10.2.2.1 Instruction Word Definition	
Bit	Description
1:0	Opcode 00 Nop 01 Move 10 Hash 11 Done
*	Source Address * uses PAR_PIB_AWIDTH
*	Destination Address * uses PAR_POB_AWIDTH
*	Length * uses PAR_SL_LEN_WIDTH

10.2.3 Implementation Information

The Slicer contains a byte wise barrel shifter that is used to pack data into the flow key. A Moore finite state machine controls the execution of commands. The command comes into the Slicer and is shifted to provide an address. The Slicer uses this address to read the Slicer Instruction Database.

10.2.4 File Names

Top: Slicer.v(hd)

Uses: ParserConstants.v(hd)

10.2.5 Pin Descriptions

10.2.5.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
SIDone	OUT	1	Slicer Done. This output is used to tell the rest of the Parser that the Slicer has finished processing the current packet.
ParserEn	IN	1	Parser Enable bit from control register

10.2.5.2 Parser Input Buffer Interface			
Signal	Dir	Width	Description
SIPIBAdd	OUT	*	Slicer Parser Input Buffer Address bus. * Uses PAR_PIB_AWIDTH
PIBuSIData	IN	*	Parser Input Buffer to Slicer Data bus. * Uses PAR_PIB_DWIDTH

10.2.5.3 Parser Output Buffer Interface			
Signal	Dir	Width	Description
SIPOBAdd	OUT	*	Slicer Parser Output Buffer Address bus. * Uses PAR_POB_AWIDTH
SIWrStb	OUT	1	Slicer Write Strobe.
SIPOBData	OUT	*	Slicer to Parser Output Buffer Data bus. * Uses PAR_POB_DWIDTH

10.2.5.4 CPU Interface MUX Interface			
Signal	Dir	Width	Description
SiAdd	OUT	*	Slicer Address bus * uses PAR_SID_AWIDTH
CMCoSIData	IN	*	CMC to Slicer Data bus * uses PAR_SID_DWIDTH

10.2.5.5 Pattern Recognition Engine Interface			
Signal	Dir	Width	Description
PREnSIEn	IN	1	Pattern Recognition Engine to Slicer Enable
PREDone	IN	1	Pattern Recognition Engine Done.
PREnSICommand	IN	*	Pattern Recognition Engine to Slicer Command bus * uses PAR_PRE_COM_WIDTH
PREnSIProtocol	IN	*	Pattern Recognition Engine to Slicer Protocol bus

			* uses PAR_PRE_PRO_WIDTH
--	--	--	--------------------------



10.2.5.6 Analyzer Interface Control Interface			
Signal	Dir	Width	Description
SIFlowKeySize	OUT	*	Pattern Recognition Engine to Slicer Protocol bus * uses PAR_SL_FKS_WIDTH

10.2.6 Verilog Module

```

/*
Slicer.v
Slicer Module

*/
#include "ParserConstants.v"

module Slicer(Reset_N, MCLK, ParserEn, SIDone, SIPIBAdd, PIBuSIData, SIPOBAdd, SIWrStb,
SIPOBData, SIAdd, CMCoSIData, PREnSIEn, PREDone, PREnSICommand, PREnSIProtocol,
SIFlowKeySize);

// General Interface Interface
input Reset_N;
input MCLK;
input ParserEn;
output SIDone;
// Parser Input Buffer Interface
output ['PAR_PIB_AWIDTH-1 : 0] SIPIBAdd;
input ['PAR_PIB_DWIDTH-1 : 0] PIBuSIData;
// Parser Output Buffer Interface
output ['PAR_POB_AWIDTH-1 : 0] SIPOBAdd;
output SIWrStb;
output ['PAR_POB_DWIDTH-1 : 0] SIPOBData;
// CMC Interface
output ['PAR_SID_AWIDTH-1 : 0] SIAdd;
output ['PAR_SID_DWIDTH-1 : 0] CMCoSIData;
// Pattern Recognition Engine Interface
input PREnSIEn;
input PREDone;
input ['PAR_PRE_COM_WIDTH-1 : 0] PREnSICommand;
input ['PAR_PRE_PRO_WIDTH-1 : 0] PREnSIProtocol;
// AIC
output ['PAR_SL_FKS_WIDTH-1 : 0] SIFlowKeySize;

```

10.3 Pattern Recognition Database Sub-module - PRD

10.3.1 Symbol

10.3.2 Highlights

- Scaleable implementation
- Wraps either RAM or ROM instantiation or can be synthesized latches

10.3.3 Description

The Pattern Recognition Database Memory module is a wrapper for the storage medium used to hold the pattern recognition database. Only the CPU can write this memory.

10.3.4 Implementation Information

The module can be synthesized or a RAM or ROM cell can be instantiated into the wrapper.

10.3.5 File Names

Top: PRD.v(hd)

Uses: ParserConstants.v(hd), GenericRAM.v(hd)

10.3.6 Pin Descriptions

10.3.6.1 CPU Interface MUX Interface			
Signal	Dir	Width	Description
CMCoPRDWr	IN	1	CMC to PRD Write Strobe
CMCoPRDAdd	IN	*	CMC to PRD Address bus * uses PAR_PRD_AWIDTH
PRDDData	OUT	*	PRD Data bus * uses PAR_PRD_DWIDTH
CMCoPRDDData	IN	*	CMC to PRD Data bus * uses PAR_PRD_DWIDTH

10.3.7 Verilog Module

```

/*
PRD.v

*/
#include "ParserConstants.v"

module PRD(CMCoPRDDData, PRDDData, CMCoPRDAdd, CMCoPRDWr);

input ['PAR_PRD_AWIDTH-1 : 0] CMCoPRDAdd;
input ['PAR_PRD_DWIDTH-1 : 0] CMCoPRDDData;
output ['PAR_PRD_DWIDTH-1 : 0] PRDDData;
input CMCoPRDWr;

```

10.4 Slicer Instruction Database Sub-module -SID

10.4.1 Symbol

10.4.2 Highlights

- Scaleable implementation
- Wraps either RAM or ROM instantiation or can be synthesized latches

10.4.3 Description

The Slicer Instruction Database module is a wrapper for the storage medium used to hold the pattern recognition database. Only the CPU can write this memory.

10.4.4 Implementation Information

The module can be synthesized or a RAM or ROM cell can be instantiated into the wrapper.

10.4.5 File Names

Top: SID.v(hd)

Uses: ParserConstants.v(hd), GenericRAM.v(hd)

10.4.6 Pin Descriptions

10.4.6.1 CPU Interface MUX Interface			
Signal	Dir	Width	Description
CMCoSIDWr	IN	1	CMC to SID Write Strobe
CMCoSIDAdd	IN	*	CMC to SID Address bus * uses PAR_SID_AWIDTH
SIDData	OUT	*	SID Data bus * uses PAR_SID_DWIDTH
CMCoSIDData	IN	*	CMC to SID Data bus * uses PAR_SID_DWIDTH

10.4.7 Verilog Module

```
/*
SID.v
```

```
*/
#include "ParserConstants.v"
```

```
module SID(CMCoSIDData, SIDData, CMCoSIDAdd, CMCoSIDWr);
```

```
input ['PAR_SID_AWIDTH-1 : 0] CMCoSIDAdd;
input ['PAR_SID_DWIDTH-1 : 0] CMCoSIDData;
output ['PAR_SID_DWIDTH-1 : 0] SIDData;
```

input CMCosIDWr;



10.5 CPU Interface MUX and Control Register Sub-module - CMC

10.5.1 Symbol

10.5.2 Description

The CPU Interface MUX and Control Register module controls the communication between the external CPU and the Parser. The CMC contains a MUX for the CPU read back. It also contains the control register for the Parser.

10.5.3 File Names

Top: CMC.v(hd)

Uses: ParserConstants.v(hd)

10.5.4 Pin Descriptions

10.5.4.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
ParserEn	OUT	1	Parser Enable bit from control register. When this bit becomes active

10.5.4.2 Slicer Instruction Database Interface			
Signal	Dir	Width	Description
CMCoSIDWr	OUT	1	CMC to SID Write Strobe
CMCoSIDAdd	OUT	*	CMC to SID Address bus * uses PAR_SID_AWIDTH
SIDData	IN	*	SID Data bus * uses PAR_SID_DWIDTH
CMCoSIDData	OUT	*	CMC to SID Data bus * uses PAR_SID_DWIDTH

10.5.4.3 Pattern Recognition Database Interface			
Signal	Dir	Width	Description
CMCoPRDWr	OUT	1	CMC to PRD Write Strobe
CMCoPRDAdd	OUT	*	CMC to PRD Address bus * uses PAR_PRD_AWIDTH
PRDData	IN	*	PRD Data bus * uses PAR_PRD_DWIDTH
CMCoPRDData	OUT	*	CMC to PRD Data bus * uses PAR_PRD_DWIDTH

10.5.4.4 Slicer Interface

Signal	Dir	Width	Description
SiAdd	IN	*	Slicer Address bus * uses PAR_SID_AWIDTH
COCoSIData	OUT	*	CMC to Slicer Data bus * uses PAR_SID_DWIDTH

10.5.4.5 Pattern Recognition Engine Interface

Signal	Dir	Width	Description
PREAdd	IN	*	Pattern Recognition Engine Address bus * uses PAR_PRD_AWIDTH
BaseOffset	OUT	4	Base Offset. This is the first offset the Pattern Recognition Engine will check.
CMCoPREData	OUT	*	CMC to Pattern Recognition Engine Data bus * uses PAR_PRD_DWIDTH

10.5.4.6 Host Interface Signals

Signal	Dir	Width	Description
ParserSel_N	IN	1	Parser Select - active low. ParserSel_N is sampled on the rising edge of MCLK . If it is active, it signifies that the external host is attempting to access the parser.
HostWrite	IN	1	Write. Write is sampled on the rising edge of MCLK . This signal is only valid when ParserSel_N is active. If this signal is active, the host is attempting to write to the parser. Inactive this signal signifies a read from the parser.
HostBlast_N	IN	1	Burst Last - active low. HostBlast_N is sampled on the rising edge of MCLK . HostBlast_N tells the parser that the current transfer is the last transfer in this burst.
HostWait_N	IN	1	Wait - active low. HostWait_N is sampled on the rising edge of MCLK . The host asserts HostWait_N when it wishes to slow transfers between itself and the parser.
ParHostReady_N	OUT	1	Parser to Host Ready - active low. ParHostReady_N should be sampled on the rising edge of MCLK . The parser returns ParHostReady_N when the current cycle is completed. For a write operation, ParHostReady_N means that the HostDataIn bus has been latched. For a read operation ParHostReady_N means that the requested data is on the ParHostDataOut bus and is valid. ParHostReady_N is blocked by HostWait_N .

HostAddress	IN	13	Host Address bus. HostAddress is sampled on the rising edge of MCLK if ParserSel_N is active. This bus defines the first address in this burst to access in the 64 Kilobyte address space of the Parser. See Section x.x.x for the Address Utilization Map.
HostByteEn_N	IN	8	Host Byte Enable bus – Active low. HostWait_N is sampled on the rising edge of MCLK.
HostDataIn	IN	64	Host Data Input bus. HostDataIn is sampled on the rising edge of MCLK if HostWrite is active and HostWait_N is inactive.
ParHostDataOut	OUT	64	ParserHost Data Output bus. ParHostDataOut should be sampled on the rising edge of MCLK. Data on this bus is valid during a read cycle when ParHostReady_N is active.

10.5.5 Verilog Module

```

/*
CMC.v
CMC - CPU Interface MUX and Control Register Module

*/
`include "ParserConstants.v"

module CMC(Reset_N, MCLK ,ParserEn, CMCoSIDWr, CMCoSIDAdd, SIDData, CMCoSIDData,
CMCoPRDWr, CMCoPRDAdd, PRDDData, CMCoPRDDData, SIAdd, CMCoSIDData, PREAdd, BaseOffset,
CMCoPREData, ParserSel_N, HostWrite, HostBlast_N, HostWait_N, ParHostReady_N, HostAddress,
HostDataIn, ParHostDataOut);

// General Interface Interface
input Reset_N;
input MCLK;
output ParserEn;
// Sicer Instruction Database Interface
output CMCoSIDWr;
output [^PAR_SID_AWIDTH-1 : 0] CMCoSIDAdd;
input [^PAR_SID_DWIDTH-1 : 0] SIDData;
output [^PAR_SID_AWIDTH-1 : 0] CMCoSIDData;
// Pattern Recognition Database Interface
output CMCoPRDWr;
output [^PAR_PRD_AWIDTH-1 : 0] CMCoPRDAdd;
input [^PAR_PRD_DWIDTH-1 : 0] PRDDData;
output [^PAR_PRD_DWIDTH-1 : 0] CMCoPRDDData;
// Slicer Interface
input [^PAR_SID_AWIDTH-1 : 0] SIAdd;
output [^PAR_SID_DWIDTH-1 : 0] CMCoSIDData;
// Pattern Recognition Engine Interface
input [^PAR_PRD_AWIDTH-1 : 0] PREAdd;
output [^PAR_BASE_OFF_WIDTH-1 : 0] BaseOffset;
output [^PAR_PRD_DWIDTH-1 : 0] CMCoPREData;
//Host Interface
input ParserSel_N;
input HostWrite;

```

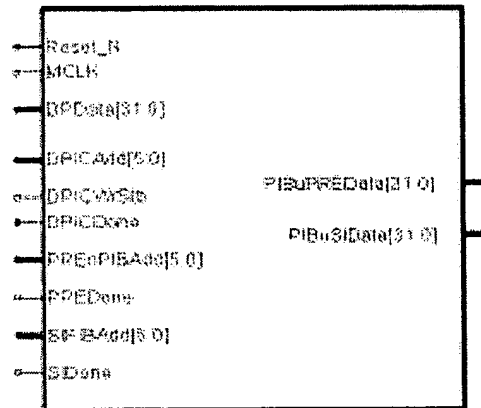


input HostBlast_N;
input HostWait_N;
output ParHostReady_N;
input ['PAR_HOST_AWIDTH-1 : 0] HostAddress;
input ['PAR_HOST_DWIDTH-1 : 0] HostDataIn;
output ['PAR_HOST_DWIDTH-1 : 0] ParHostDataOut;



10.6 Parser Input Buffer Sub-module – PIB

10.6.1 Symbol



10.6.2 Highlights

- Scaleable implementation
- Asynchronous three ported RAM
- Can be build from three separate single port RAM cells
- Wraps either RAM instantiation or can be synthesized latches
- Separate dual read and a single write interfaces

10.6.3 Description

The Parser Input Buffer is a wrapper for the buffer that is used to store the start of the packet. It is three ported with separate dual read and a single write interfaces. The data from the DataPort interface is stored in one of three logical or physical buffers through the write port. The Pattern Recognition Engine uses one of the read ports and the Slicer uses the other. The three interfaces never access the same third of the buffer at the same time. Each of the interfaces looks like a single buffer to the attached modules. The Parser Input Buffer controls which of the three buffers the module is controlling. When the first packet comes in the DataPort Interface Control module writes the data into one of the three buffers. It then increments a modulo three counter to point to the next buffer. The Pattern Recognition Engine will then begin processing the packet. Finally after the Pattern Recognition Engine is finished the Slicer will get access to the buffer. In this way each of the three processes have access to a buffer and each get access to the packet in turn.

10.6.4 Implementation Information

The module can be synthesized or RAM cells can be instantiated into the wrapper. The instantiated RAM can be either a single three ported cell or three separate RAM cells. The Parser Input Buffer can be three separate RAM cells because the control logic will never try to read and write the same third of the buffer at the same time.

10.6.5 File Names

Top: PIB.v(hd)

Uses: ParserConstants.v(hd), Generic3PortRAM.v(hd)

10.6.6 Pin Descriptions

10.6.6.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
ParserEn	IN	1	Parser Enable bit from control register

10.6.6.2 DataPort Interface			
Signal	Dir	Width	Description
DPData	IN	*	DataPort Data bus. * Uses PAR_DP_DWIDTH

10.6.6.3 DataPort Interface Control Interface			
Signal	Dir	Width	Description
DPICAdd	IN	*	DataPort Interface Control Address bus. * Uses PAR_PIB_AWIDTH
DPICDone	IN	1	DataPort Interface Control Done. This input is used to tell the Parser Input Buffer that the DataPort Interface Control module has finished writing the buffer. The Parser Input Buffer also uses this signal to increment it's internal pointer so that the next address from the DataPort Interface Control will point to the next packet buffer. DPICAdd is ignored for one cycle after DPICDone is active.
DPICWriteStb	IN	1	DataPort Interface Control Write Strobe.

10.6.6.4 Pattern Recognition Engine Interface			
Signal	Dir	Width	Description
PREnPIBAdd	IN	*	Pattern Recognition Engine Parser Input Buffer Address bus. * Uses PAR_PIB_AWIDTH

10.6.6.4 Pattern Recognition Engine Interface

Signal	Dir	Width	Description
PREDone	IN	1	Pattern Recognition Engine Done. This input is used to tell the Parser Input Buffer that the Pattern Recognition Engine has finished processing the current packet and the buffer can be freed. The Parser Input Buffer also uses this signal to increment it's internal pointer so that the next address from the Pattern Recognition Engine will point to the next packet buffer. PREnPIBAdd is ignored for one cycle after PREDone is active.
PIBuPREData	OUT	*	Parser Input Buffer to Pattern Recognition Engine Data bus. * Uses PAR_PIB_DWIDTH

10.6.6.5 Slicer Interface

Signal	Dir	Width	Description
SIPIBAdd	IN	*	Slicer Parser Input Buffer Address bus. * Uses PAR_PIB_AWIDTH
SIDone	IN	1	Slicer Done. This input is used to tell the Parser Input Buffer that the Slicer has finished processing the current packet and the buffer can be freed. The Parser Input Buffer also uses this signal to increment it's internal pointer so that the next address from the Slicer will point to the next packet buffer. SIPIBAdd is ignored for one cycle after SIDone is active.
PIBuSiData	OUT	*	Parser Input Buffer to Slicer Data bus. * Uses PAR_PIB_DWIDTH

10.6.7 Verilog Module

```
/*
PIB.v
```

```
*/
`include "ParserConstants.v"
```

```
module PIB(Reset_N, MCLK ,ParserEn, DPData, DPICAdd, DPICDone, DPICWrStb, PREnPIBAdd,
PREDone, PIBuPREData, SIPIBAdd, SIDone, PIBuSiData);
```

```
input Reset_N;
input MCLK;
input ParserEn;
input DPICDone;
input DPICWrStb;
input PREDone;
input SIDone;
input [`PAR_PIB_DWIDTH-1 : 0] DPData;
input [`PAR_PIB_AWIDTH-1 : 0] DPICAdd;
input [`PAR_PIB_AWIDTH-1 : 0] PREnPIBAdd;
```

input ['PAR_PIB_AWIDTH-1 : 0] SIPIBAdd;
output ['PAR_PIB_DWIDTH-1 : 0] PIBuPREData;
output ['PAR_PIB_DWIDTH-1 : 0] PIBuSIData;



10.7 Parser Output Buffer Sub-module - POB

10.7.1 Symbol

10.7.2 Highlights

- Scalable implementation
- Asynchronous dual ported RAM
- Can be build from two separate single port RAM cells
- Wraps either RAM instantiation or can be synthesized latches
- Separate read and write interfaces

10.7.3 Description

The Parser Output Buffer is a wrapper for the buffer that is used to store the output of the Slicer. It is dual ported with separate read and write interfaces. The write interface is controlled by the Slicer. The read interface is controlled by the Analyzer Interface Control logic. The Parser Output Buffer maintains a pointer to the two buffers such that one buffer is controlled by the Slicer and one is controlled by the Analyzer Interface Control logic.

10.7.4 Implementation Information

The module can be synthesized or RAM cells can be instantiated into the wrapper. The instantiated RAM can be either a single dual ported cell or two separate RAM cells. The Parser Output Buffer can be two separate RAM cells because the control logic will never try to read and write the same half of the buffer at the same time.

10.7.5 File Names

Top: POB.v(hd)

Uses: ParserConstants.v(hd), Generic2PortRAM.v(hd)

10.7.6 Pin Descriptions

10.7.6.1 General Interface Signals

Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
ParserEn	IN	1	Parser Enable bit from control register

10.7.6.2 Slicer Interface

Signal	Dir	Width	Description
SIPOBAdd	IN	*	Slicer Parser Output Buffer Address bus. * Uses PAR_POB_AWIDTH
SIDone	IN	1	Slicer Done. This input is used to tell the Parser Output Buffer that the Slicer has finished processing the current flow and the buffer can be sent to the Analyzer. The Parser Output Buffer also uses this signal to increment it's internal pointer so that the next address from the Slicer will point to the next flow buffer. SIPOBAdd is ignored for one cycle after SIDone is active.
SIWrStb	IN	1	Slicer Write Strobe.
SIPOBData	IN	*	Slicer to Parser Output Buffer Data bus. * Uses PAR_POB_DWIDTH

10.7.6.3 Analyzer Interface Control Interface

Signal	Dir	Width	Description
AICoPOBAdd	IN	*	Analyzer Interface Control to Parser Output Buffer Address bus. * Uses PAR_POB_AWIDTH
AICDone	IN	1	Analyzer Interface Control Done. This input is used to tell the Parser Output Buffer that the Analyzer Interface Control has finished sending the current flow to the Analyzer. The Parser Output Buffer also uses this signal to increment it's internal pointer so that the next address from the Analyzer Interface Control will point to the next flow buffer. AICoPOBAdd is ignored for one cycle after AICDone is active.

10.7.6.4 Analyzer Interface

Signal	Dir	Width	Description
ParserData	OUT	*	Parser Data bus. * Uses PAR_ANA_DWIDTH

10.7.7 Verilog Module

```
/*
POB.v

*/
#include "ParserConstants.v"

module POB(Reset_N, MCLK ,ParserEn, SIPOBData, SIPOBAdd, SIDone, SIWrStb,
           AICoPOBAdd, AICDone, ParserData);

input Reset_N;
input MCLK;
input ParserEn;
input SIDone;
input SIWrStb;
input AICDone;
input ['PAR_POB_DWIDTH-1 : 0] SIPOBData;
input ['PAR_POB_AWIDTH-1 : 0] SIPOBAdd;
input ['PAR_POB_AWIDTH-1 : 0] AICoPOBAdd;
output ['PAR_POB_DWIDTH-1 : 0] ParserData;
```

10.8 DataPort Interface Control Sub-module - DPIC

10.8.1 Symbol

10.8.2 Description

The DataPort Interface Control module handshakes with the external source of packets. The external device starts sending the packet to the DataPort Interface Control module by asserting **DPPacketDelim**. The transfer of data is coordinated by the **DPDataStb_N/DPReady_N** pair. If the external device decides to about the packet it can assert **DPKillPkt_N**.

10.8.3 Implementation Information

The Analyzer Interface Control module is implemented as a Moore type finite state machine. Each of the outputs of the state machine are registered to assure maximum setup time for the external device.

10.8.4 File Names

Top: DPIC.v(hd)

Uses: ParserConstants.v(hd)

10.8.5 Pin Descriptions

10.8.5.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
ParserEn	IN	1	Parser Enable bit from control register

10.8.5.2 DataPort Interface			
Signal	Dir	Width	Description
DPPacketDelim	IN	1	DataPort Packet Delimiter. This signal should be driven active when the external logic wants to send a packet to the parser. DPPacketDelim should remain active during the entire packet transfer. DPPacketDelim must go inactive for one clock between packets.
DPDataStb_N	IN	1	DataPort Data Strobe. When active, this signal tells the parser that data on the DPData bus is valid. If DPReady_N was inactive at the end of the previous cycle, DPDataStb_N should not be driven active. If DPReady_N goes inactive in the same cycle as DPDataStb_N , then the parser will latch the incoming data so that no data is lost.

10.8.5.2 DataPort Interface

Signal	Dir	Width	Description
DPKillPkt_N	IN	1	DataPort Kill Packet. If this signal becomes active while DPPacketDelim is active, the parser will attempt to stop processing the current packet and flush it's input Buffer. If however, parsing of the packet is completed, the packet will not be able to be recalled. This should only be a problem in a 'cut through' implementation.
DPReady_N	OUT	1	DataPort Ready – active low. This signal when driven active means that the parser can accept new data. If however the parser's input Buffer is filled, DPReady_N will be driven inactive. To prevent overruns, DPReady_N will go inactive when the parser can actually accept one more data transfer.

10.8.5.3 Parser Input Buffer Interface

Signal	Dir	Width	Description
DPICAdd	OUT	*	DataPort Interface Control Address bus. * Uses PAR_PIB_AWIDTH
DPICDone	OUT	1	DataPort Interface Control Done. This output is used to tell the Parser Input Buffer that the DataPort Interface Control module has finished writing the buffer.
DPICWriteStb	OUT	1	DataPort Interface Control Write Strobe.

10.8.5.4 Pattern Recognition Engine Interface

Signal	Dir	Width	Description
PREDone	IN	1	Pattern Recognition Engine Done

10.8.6 Verilog Module

/*

DPIC.v

*/

`include "ParserConstants.v"

```
module DPIC(Reset_N, MCLK ,ParserEn, DPPacketDelim, DPDataStb, DPKillPkt_N, DPReady_N,
DPICAdd, DPICDone, DPICWrStb, PREDone);
```

```
input Reset_N;
input MCLK;
input ParserEn;
input DPPacketDelim;
```

```
input DPDataStb;  
input DPKillPkt_N;  
input PREDone;  
output DPReady_N;  
output DPICDone;  
output DPICWrStb;  
output ['PAR_PIB_AWIDTH-1 : 0] DPICAdd;
```



10.9 Analyzer Interface Control Sub-module -AIC

10.9.1 Symbol

10.9.2 Description

The Analyzer Interface Control module handshakes with the Analyzer in order to transfer the flow key for further processing. The Analyzer Interface Control module starts a transfer to the Analyzer by asserting **ParserKeyDelim**. It then transfers the data via the **AnalyzerReady/ParserDataAvail** handshake pair. The Analyzer Interface Control module also sends the address of the data to be sent to the Parser Output Buffer.

10.9.3 Implementation Information

The Analyzer Interface Control module is implemented as a Moore type finite state machine. Each of the outputs of the state machine are registered to assure maximum setup time for the Analyzer interface.

10.9.4 File Names

Top: AIC.v(hd)

Uses: ParserConstants.v(hd)

10.9.5 Pin Descriptions

10.9.5.1 General Interface Signals

Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
ParserEn	IN	1	Parser Enable bit from control register

10.9.5.2 Analyzer Interface

Signal	Dir	Width	Description
AnalyzerReady	IN	1	Analyzer Ready. This signal tells the parser that the analyzer can accept data.
ParserKeyDelim	OUT	1	Parser Key Delimiter. The ParserKeyDelim signal becomes active when the first quadword of a new key is ready to transfer to the analyzer. It goes inactive when the last quadword of the key is transferred.
ParserDataAvail	OUT	1	Parser Data Available. If this signal is active the data on the ParserData bus is valid.

10.9.5.3 Slicer Interface

Signal	Dir	Width	Description
SIFlowKeySize	IN	*	Slicer Flow Key Size bus. This bus is valid when SlDone is active. It communicates the size of the flow key so the Analyzer Interface Control can send the right amount of data to the Analyzer. * uses PAR_MAX_FLOW_KEY_SIZE
SlDone	IN	1	Slicer Done. This input is used to tell the Analyzer Interface Control that the Slicer has finished processing the current packet and can be sent to the Analyzer.

10.9.5.4 Parser Output Buffer Interface

Signal	Dir	Width	Description
AICoPOBAdd	OUT	*	Analyzer Interface Control to Parser Output Buffer Address bus. * Uses PAR_POB_AWIDTH
AICDone	OUT	1	Analyzer Interface Control Done. This output is used to tell the Parser Output Buffer that the Analyzer Interface Control has finished sending the current flow to the Analyzer.

10.9.6 Verilog Module

/*

AIC.v

*/

'include "ParserConstants.v"

```
module AIC(Reset_N, MCLK, ParserEn, AnalyzerReady, ParserKeyDelim, ParserDataAvail,
SIFlowKeySize, SlDone, AICoPOBAdd, AICDone);
```

```
input Reset_N;
input MCLK;
input ParserEn;
input AnalyzerReady;
output ParserKeyDelim;
output ParserDataAvail;
input SlDone;
input ['PAR_SL_FKS_WIDTH-1 : 0]SIFlowKeySize;
output ['PAR_PIB_AWIDTH-1 : 0] AICoPOBAdd;
output AICDone;
```




- **Exhibit A3:** Technically Elite MeterFlow Accelerator Analyzer Module Specification (Document MFAAnalyze.pdf)

Technically Elite MeterFlow Accelerator Analyzer Module Specification

Not For External Release!

Revision History		
Version	Date	Description
0.2	[REDACTED]	
0.9	[REDACTED]	Final Prerelease

0 Table of Contents

0	Table of Contents	2
1	Introduction.....	5
2	Technically Elite MeterFlow Accelerator Analyzer Module Highlights	5
3	Architectural Overview	6
3.1	Flow Database.....	7
3.1.1	Extracted Input Data from Parser Diagram	8
3.1.2	Flow Entry Description	9
3.2	Architectural Block Diagram	9
4	Top Level MeterFlow Accelerator Analyzer Module Symbol	10
5	MeterFlow Accelerator Analyzer Module Top Level Pin Descriptions	11
5.1.1.1	General Interface Signals	11
5.1.1.2	Memory Interface	11
5.1.1.3	Host Interface Signals	12
5.1.1.4	Parser Interface	13
5.1.1.5	Known Flow Interface.....	13
6	MeterFlow Accelerator Analyzer Module Top Level VHDL Entity	14
7	MeterFlow Accelerator Analyzer Module Top Level Verilog Module	14
8	MeterFlow Accelerator Analyzer Module Top Level Schematic.....	14
9	Analyzer Module Constants Files	15
9.1	Analyzer module Verilog Constants File – ParserConstants.v	15
9.2	Analyzer module VHDL Constants File – ParserConstants.vhd	15
10	Sub-module Descriptions.....	16
10.1	Unified Flow Key Buffer - UFKB.....	16
10.1.1	Symbol	16
10.1.2	Highlights.....	16
10.1.3	Description.....	16
10.1.4	Implementation Information.....	16
10.1.5	File Names	16
10.1.6	Pin Descriptions	16
10.1.6.1	General Interface Signals	16
10.1.6.2	Parser Interface	17
10.1.6.3	Lookup and Update Engine Interface.....	17
10.1.6.4	State Processor Interface.....	18
10.1.6.5	Flow Insertion and Deletion Engine Interface.....	18
10.1.7	Verilog Module	19
10.1.8	VHDL Component	20
10.2	Lookup and Update Engine - LUE.....	21
10.2.1	Symbol	21
10.2.2	Highlights.....	21
10.2.3	Description.....	21
10.2.4	Implementation Information.....	21
10.2.5	File Names	21
10.2.6	Pin Descriptions	21
10.2.6.1	General Interface Signals	21
10.2.6.2	Unified Flow Key Buffer Interface	21
10.2.6.3	Cache Interface.....	22
10.2.6.4	Known Flow Interface.....	23
10.2.7	Verilog Module	23

10.2.8	VHDL Component	23
10.3	Analyzer CPU Interface and Control - ACIC	24
10.3.1	Symbol	24
10.3.2	Description	24
10.3.3	File Names	24
10.3.4	Pin Descriptions	24
10.3.4.1	General Interface Signals	24
10.3.4.2	Host Interface Signals	24
10.3.4.3	Cache Interface	25
10.3.4.4	State Processor Instruction Database Interface	26
10.3.5	Verilog Module	26
10.3.6	VHDL Component	26
10.4	Flow Insertion and Deletion Engine - FIDE	27
10.4.1	Symbol	27
10.4.2	Highlights	27
10.4.3	Description	27
10.4.4	Implementation Information	27
10.4.5	File Names	27
10.4.6	Pin Descriptions	27
10.4.6.1	General Interface Signals	27
10.4.6.2	Unified Flow Key Buffer Interface	27
10.4.6.3	Cache Interface	28
10.4.7	Verilog Module	28
10.4.8	VHDL Component	29
10.5	State Processor Instruction Database - SPID	30
10.5.1	Symbol	30
10.5.2	Highlights	30
10.5.3	Description	30
10.5.4	Implementation Information	30
10.5.5	File Names	30
10.5.6	Pin Descriptions	30
10.5.6.1	General Interface Signals	30
10.5.6.2	Analyzer CPU Interface Control Interface	30
10.5.6.3	State Processor Interface	31
10.5.7	Verilog Module	31
10.5.8	VHDL Component	31
10.6	Unified Memory Controller - UMC	32
10.6.1	Symbol	32
10.6.2	Highlights	32
10.6.3	Description	32
10.6.4	Implementation Information	32
10.6.5	File Names	32
10.6.6	Pin Descriptions	32
10.6.6.1	General Interface Signals	32
10.6.6.2	Memory Interface	32
10.6.6.3	Cache Interface	33
10.6.7	Verilog Module	33
10.6.8	VHDL Component	34
10.7	Cache	35
10.7.1	Symbol	35
10.7.2	Highlights	35
10.7.3	Description	35
10.7.3.1	Priority	35
10.7.4	Implementation Information	36
10.7.5	File Names	36

10.7.6	Pin Descriptions	36
10.7.6.1	General Interface Signals	36
10.7.6.2	Unified Memory Controller Interface.....	36
10.7.6.3	Flow Insertion and Deletion Engine Interface.....	36
10.7.6.4	Analyzer CPU Interface Control Interface	37
10.7.6.5	Lookup Engine Interface	37
10.7.6.6	State Processor Interface	38
10.7.7	Verilog Module	38
10.7.8	VHDL Component	39
10.8	State Processor - SP	40
10.8.1	Symbol	40
10.8.2	Highlights	40
10.8.3	Description	40
10.8.4	Architecture	40
10.8.4.1	Scratch Pad Registers	40
10.8.4.2	Instruction Pointer and Stack	40
10.8.4.3	Flag Register	40
10.8.4.3.1	Flag Register Word Definition.....	40
10.8.4.4	Compare Block	41
10.8.4.5	Flow Key Pointer	41
10.8.4.6	Flow Entry Pointer	41
10.8.5	Instruction Definitions.....	41
10.8.5.1	Jump	41
10.8.5.2	Call	41
10.8.5.3	Return.....	41
10.8.5.4	Copy	42
10.8.5.5	Compare	42
10.8.5.6	Instruction Word Definition	42
10.8.6	Implementation Information.....	42
10.8.7	File Names	42
10.8.8	Pin Descriptions	42
10.8.8.1	General Interface Signals	42
10.8.8.2	Unified Flow Key Buffer Interface	42
10.8.8.3	Cache Interface.....	43
10.8.8.4	State Processor Interface	43
10.8.9	Verilog Module	44
10.8.10	VHDL Component	44
11	Appendix A - Multi-Packet State Processing.....	45
11.1	Overview	45
11.2	Analyzer Data Input Requirements	45
11.3	State-base Traffic Classification	45
11.3.1	Session Tracking	45
11.3.2	Server Announcement	46
11.3.2.1	Sun RPC Analysis	46
11.3.2.2	Process for Sun RPC Analysis	47
11.3.3	Port Mapper Operation	50
11.3.4	Service Announcement.....	50
11.3.5	In-stream Recognition and Extraction.....	50
11.3.5.1	Web-based Applications	50

1 Introduction

This document is designed to be the repository for all information related to the MeterFlow Accelerator Analyzer Module. This specification is designed to provide the engineer with enough information to fully implement the module. There will be revisions during and after the implementation process that will be reflected in this document.

Each part of this specification describes a different aspect of the module. It concentrates on the interfaces between the analyzer module and the other parts of the system. The other parts of the system include the parser module, the host interface module and importantly the software that models, programs and tests the system. The key to a successful implementation is the interfaces between modules and between sub-module and sub-module. Each interface is described in detail. Any changes to the interfaces may affect the entire module and even the entire system. Care must be taken that each interface is understood completely before implementation is begun.

2 Technically Elite MeterFlow Accelerator Analyzer Module Highlights

- Flexible Rule-based Traffic Classification
- State-based Tracking of Traffic
- *Multiple* Packets for Layer Processing
- Internal Cache and Memory Controller
- Direct High Bandwidth (64 bit) Memory Interface
- SG/SDRAM Support
- Programmable Rules/State Processor
- Selectable Protocols in Flows
- Future Protocols Support
- Scalable System Design

3 Architectural Overview

The analyzer module consists five major sub-modules with several supporting sub-modules. The major sub-modules are the flow lookup/update engine, the flow insertion and deletion engine, the state processor, the cache, and the unified memory controller. Each of these sub-modules work in parallel to create and update flows.

As a flow key enters the analyzer, the lookup engine attempts to find it in the flow database. If the flow exists, the lookup engine retrieves the flow from the cache. It then makes a decision based on the state information included in the flow entry to either send it to the state processor or not. In either case it updates the flow entry. This updating consists of adding values to counters in the flow database entry. If a flow does not exist, the state processor sends the flow key to the flow insertion and deletion engine which adds the flow to the database.

The state processor updates the flow based on the current state and the flow key information. The state processor processes single and multi packet protocol recognition. It may have to search through a series of possible states to determine the flow's actual state. The result of the state processor's processing is a consolidated flow entry. For example, a PointCast session will open multiple conversations that on a packet by packet basis look like separate flows. Since each conversation is merely a subflow under the PointCast master flow, a single flow that consolidates all of the information for the flow is desired.

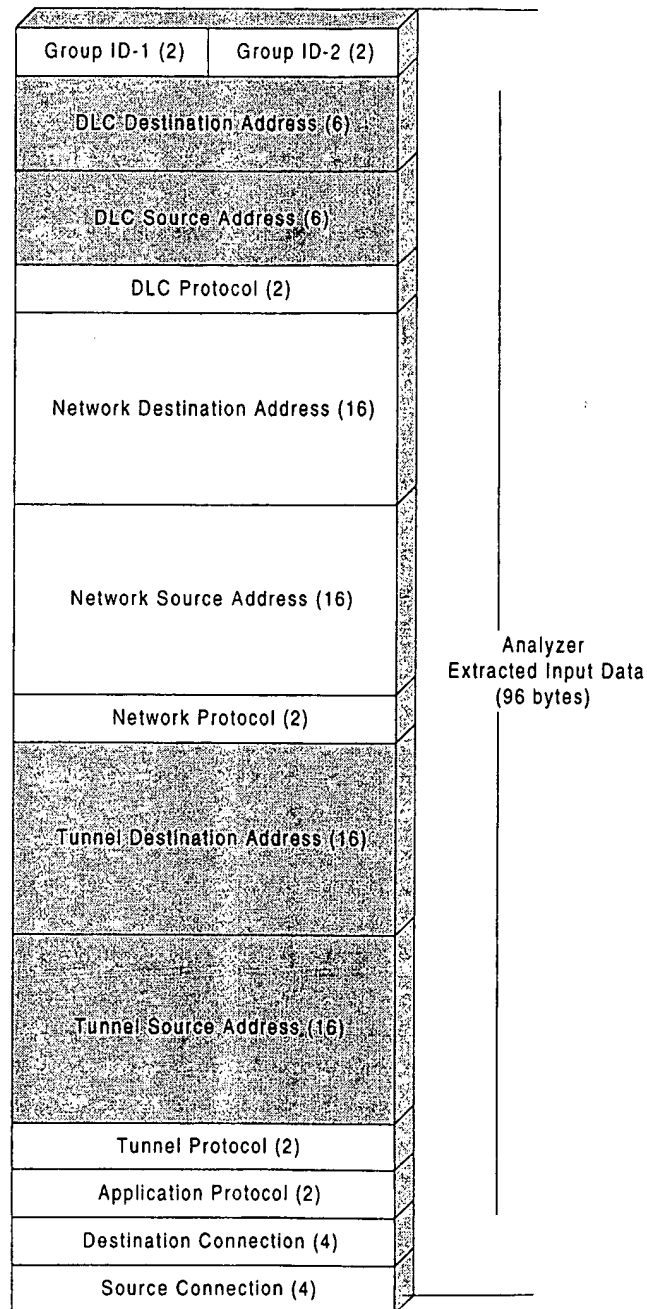
The unified memory controller can be setup to work with various configurations of SDRAM or SGRAM. It also controls the SRAM tag memory for shadowing of flow entries.

The cache is used to optimize memory bandwidth. On a typical network the packets will have a certain amount of congruity. This means that the cache can have a high hit rate with .

3.1 Flow Database

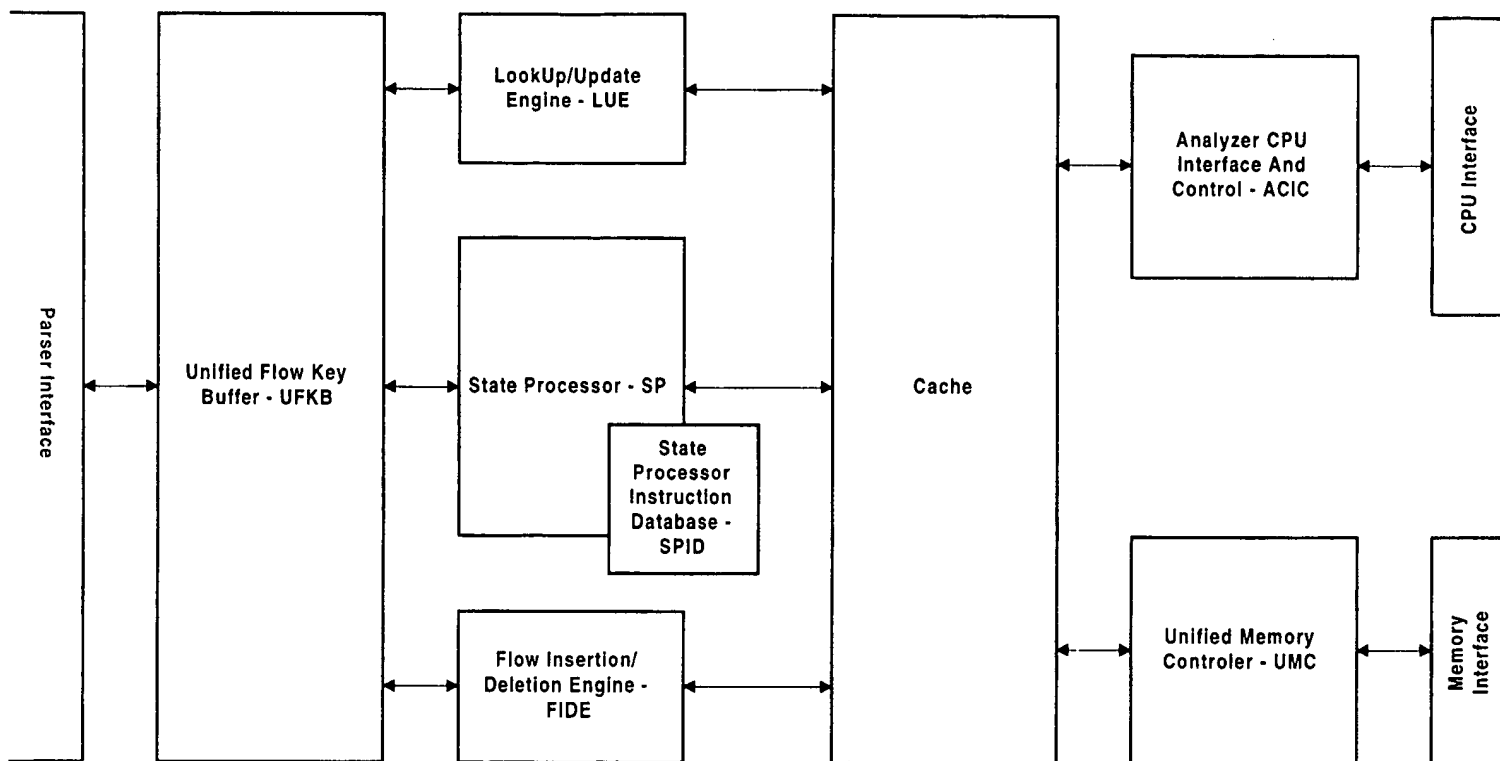
The Flow Database consists of a series of 128 byte entries. Each entry completely describes a flow. The format and information contained in the flow is described in section xxx. The database is organized into buckets. Each bucket contains n flow entries. N is determined by the designer. Buckets are accessed via a hash value created by the Parser based on information in the packet. This hash spreads the flows across the database and is based on a proprietary Technically Elite algorithm. This method allows fast look up of an entry while allowing for shallower buckets. The designer selects the bucket depth based on the amount of memory attached to the analyzer and the number of bits of the hash value used. For example, for 128k flow entries 16 Megabytes are required. Using a 16 bit hash gives two entries per bucket. This has been empirically shown to be more than adequate for the vast majority of cases.

3.1.1 Extracted Input Data from Parser Diagram

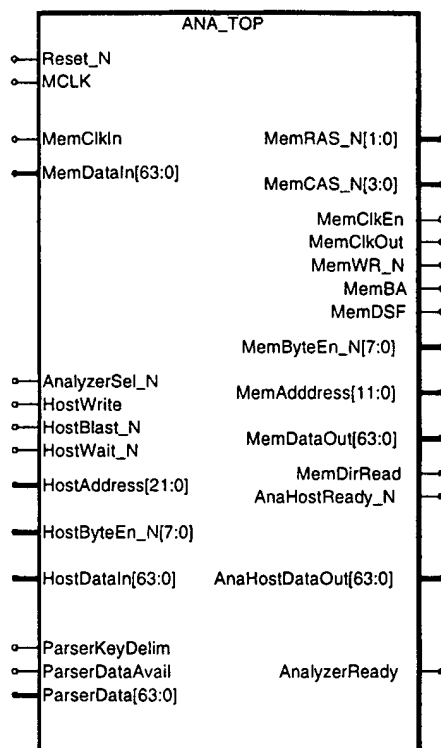


3.1.2 Flow Entry Description

3.2 Architectural Block Diagram



4 Top Level MeterFlow Accelerator Analyzer Module Symbol



5 MeterFlow Accelerator Analyzer Module Top Level Pin Descriptions

5.1.1.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low. When this signal is active the analyzer sets it's registers to their default condition and suspends operation. It will only respond to host access cycles.
MCLK	IN	1	Module Clock. All internal and external transfers except for memory transfers are synchronized by this signal.

5.1.1.2 Memory Interface			
Signal	Dir	Width	Description
MemClkIn	IN	1	Memory clock in. This signal is used to generate the memory interface timing.
MemRAS_N	OUT	*	Memory Row Address Strobe bus – active low. * uses AN_MEM_RASWIDTH
MemCAS_N	OUT	*	Memory Column Address Strobe bus– active low. * uses AN_MEM_CASWIDTH
MemClkEn	OUT	1	Memory Clock Enable. Some memories require this signal to be disabled for a certain amount of time after reset.
MemClkOut	OUT	1	Memory Clock Out. This signal is used by synchronous memory for all operations. MemClkIn is buffered and sent out on this pin. This helps reduce skew between this clock and the other signals.
MemWR_N	OUT	1	Memory Write – active low.
MemBA	OUT	1	Memory Bank Address. Used by multi-bank memory to select the bank the current operation is to operate on.
MemDSF	OUT	1	Memory Special Function select.
MemByteEn_N	OUT	*	Memory Byte Enable bus– active low. * uses AN_MEM_BEWIDTH
MemAddress	OUT	*	Memory Address bus. * uses AN_MEM_AWIDTH
MemDataIn	IN	*	Memory Data Input bus. * uses AN_MEM_DWIDTH
MemDataOut	OUT	*	Memory Data Output bus. * uses AN_MEM_DWIDTH

5.1.1.2 Memory Interface

Signal	Dir	Width	Description
MemDirRead	OUT	1	Memory Data bus Direction is Read. This signal is used to control the tri-state enable on the bidirectional memory data bus. If MemDirRead is active data is coming into the analyzer from the memory. If it is inactive the analyzer is driving data out to the memory.

5.1.1.3 Host Interface Signals

Signal	Dir	Width	Description
AnalyzerSel_N	IN	1	Host interface Analyzer Select - active low. AnalyzerSel_N is sampled on the rising edge of MCLK . If it is active, it signifies that the external host is attempting to access the analyzer.
HostWrite	IN	1	Write. Write is sampled on the rising edge of MCLK . This signal is only valid when AnalyzerSel_N is active. If this signal is active, the host is attempting to write to the analyzer. Inactive this signal sign signifies a read from the analyzer. It should also be used to control the direction of the host data bus if it is bidirectional.
HostBlast_N	IN	1	Burst Last – active low. HostBlast_N is sampled on the rising edge of MCLK . HostBlast_N tells the analyzer that the current transfer is the last transfer in this burst.
HostWait_N	IN	1	Wait – active low. HostWait_N is sampled on the rising edge of MCLK . The host asserts HostWait_N when it wishes to slow transfers between itself and the analyzer. This could also be used by additional interface logic to slow transfers so it can multiplex the bus down to a smaller size without additional FIFOs. If wait is active, HostReady_N is blocked.
AnaHostReady_N	OUT	1	Analyzer to Host Ready – active low. AnaHostReady_N should be sampled on the rising edge of MCLK . The analyzer returns AnaHostReady_N when the current cycle is completed. For a write operation, AnaHostReady_N means that the HostDataIn bus has been latched. For a read operation AnaHostReady_N means that the requested data is on the HostDataOut bus and is valid. AnaHostReady_N is blocked by HostWait_N .
HostAddress	IN	*	Host Address bus. HostAddress is sampled on the rising edge of MCLK if AnalyzerSel_N is active. This bus defines the first address in this burst to access in the 32 Megabyte address space of the analyzer. See Section x.x.x for the Address Utilization Map. * Uses AN_HOST_AWIDTH
HostByteEn_N	IN	*	Host Byte Enable bus – Active low. HostWait_N is sampled on the rising edge of MCLK . * Uses AN_HOST_BEWIDTH

5.1.1.3 Host Interface Signals

Signal	Dir	Width	Description
HostDataIn	IN	*	Host Data Input bus. HostDataIn is sampled on the rising edge of MCLK if HostWrite is active and HostWait_N is inactive. * Uses AN_HOST_DWIDTH
AnaHostDataOut	OUT	*	Analyzer Host Data Output bus. AnaHostDataOut should be sampled on the rising edge of MCLK. Data on this bus is valid during a read cycle when AnaHostReady_N is active. * Uses AN_HOST_DWIDTH

5.1.1.4 Parser Interface

Signal	Dir	Width	Description
AnalyzerReady	OUT	1	Analyzer Ready. This signal tells the parser that the analyzer can accept data.
AnalyzerAbort	OUT	1	Analyzer Abort. This signal tells the parser that the analyzer does not need any more of the flow key.
ParserKeyDelim	IN	1	Parser Key Delimiter. The ParserKeyDelim signal becomes active when the first quadword of a new key is ready to transfer to the analyzer. It goes inactive when the last quadword of the key is transferred or AnalyzerAbort is active.
ParserDataAvail	IN	1	Parser Data Available. If this signal is active the data on the ParserData bus is valid.
ParserData	IN	*	Parser Data bus.

5.1.1.5 Known Flow Interface

Signal	Dir	Width	Description
PacketRef	OUT	*	Packet Reference number bus. This bus outputs the packet reference number copied from the UFKB. * Uses AN_FR_WIDTH
Protocol	OUT	*	Protocol bus. This bus outputs the highest level protocol the State Processor has determined the packet contains. * Uses AN_APP_WIDTH
KnownFlowStb	OUT	1	Known Flow Strobe. When this signal is active, the data on the PacketRef and the Protocol busses are valid.

6 MeterFlow Accelerator Analyzer Module Top Level VHDL Entity

7 MeterFlow Accelerator Analyzer Module Top Level Verilog Module

8 MeterFlow Accelerator Analyzer Module Top Level Schematic

Insert Schematic Here



9 Analyzer Module Constants Files

The analyzer module constants files contain a list of constants used to allow rapid configuration of the module. For example the size of the Analyzer's input buffer data bus:

Verilog

```
'define AN_UFKB_DWIDTH 64 // Unified Flow Key Buffer Data Bus Width
```

VHDL

```
constant AN_UFKB_DWIDTH : integer := 64; -- Unified Flow Key Buffer Data Bus Width
```

9.1 Analyzer module Verilog Constants File – ParserConstants.v

Insert AnalyzerConstants.v here

9.2 Analyzer module VHDL Constants File – ParserConstants.vhd

Insert AnalyzerConstants.vhd here

10 Sub-module Descriptions

10.1 Unified Flow Key Buffer - UFKB

10.1.1 Symbol

10.1.2 Highlights

- Scalable implementation
- Can be build from four separate dual port RAM cells
- Wraps either RAM instantiation or can be synthesized latches
- Separate read and write interfaces

10.1.3 Description

The Unified Flow Key Buffer is a wrapper for the buffers that are used to store the flow keys from the Parser and modified flow keys from the Lookup and Update Engine and the State Processor. It is four ported with separate read and write interfaces. The four connections are to the Parser/Parser Interface Control, the Lookup and Update Engine, the State Processor and the Flow Insertion and Deletion Engine. In the Unified Flow Key Buffer logic hides from the interface which of the buffers is being accessed.

When the first word of the flow key arrives from the Parser, the Lookup and Update Engine is notified. The Lookup and Update Engine places the first address it wants on the **LUEnUFKBAdd** bus and asserts **LUEnUFKBRdReq**. If the address requested is in the buffer the Unified Flow Key Buffer asserts **UFKBuLUERdy**. If not it waits for either the data to arrive or the transfer is terminated. Once the Lookup and Update Engine finishes processing the flow key it asserts **LUEDone**. At the same time it will assert **LUEHoldBuf**. **LUEHoldBuf** tells the system that the buffer is to be sent to the State Processor.

The State Processor and Flow Insertion and Deletion Engine have similar interfaces except that the data is assumed to be already in the buffer so no ready is returned. Also Flow Insertion and Deletion Engine has no need to hold the buffer for another process so that once **FIDEDone** is asserted the buffer is freed.

10.1.4 Implementation Information

The module can be synthesized or RAM cells can be instantiated into the wrapper. The instantiated RAM should be four separate dual ported RAM cells.

The RAM must complete a write or read in a single cycle with simultaneous read and write to SEPARATE locations.

10.1.5 File Names

Top: UFKB.v(hd)

Uses: AnalyzerConstants.v(hd), Generic4PortRAM.v(hd)

10.1.6 Pin Descriptions

10.1.6.1 General Interface Signals			
Signal	Dir	Width	Description

10.1.6.1 General Interface Signals

Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
AnalyzerEn	IN	1	Analyzer Enable bit from the control register

10.1.6.2 Parser Interface

Signal	Dir	Width	Description
AnalyzerReady	OUT	1	Analyzer Ready. This signal tells the parser that the analyzer can accept data.
AnalyzerAbort	OUT	1	Analyzer Abort. This signal tells the parser that the analyzer does not need any more of the flow key. It is generated if the Lookup and Update Engine asserts LUEDone and not LUEHoldBuf before ParserKeyDelim goes inactive.
ParserKeyDelim	IN	1	Parser Key Delimiter. The ParserKeyDelim signal becomes active when the first word of a new key is ready to transfer to the analyzer. It goes inactive when the last word of the key is transferred.
ParserDataAvail	IN	1	Parser Data Available. If this signal is active, the data on the ParserData bus is valid.

10.1.6.3 Lookup and Update Engine Interface

Signal	Dir	Width	Description
UFKBuLUEDData	OUT	*	Unified Flow Key Buffer to Lookup and Update Engine read Data bus. * Uses AN_UFKB_DWIDTH
LUEnUFKBData	IN	*	Lookup and Update Engine to Unified Flow Key Buffer write Data bus. * Uses AN_UFKB_DWIDTH
LUEnUFKBAdd	IN	*	Lookup and Update Engine to Unified Flow Key Buffer Address bus. * Uses AN_UFKB_AWIDTH
FlowKeySt	OUT	1	Flow Key Start. This signal tells the Lookup and Update Engine that the Unified Flow Key Buffer module has placed the first word of a flow key buffer.
UFKBuLUERdy	OUT	1	Unified Flow Key Buffer to Lookup and Update Engine Ready.
UFKBuLUEErr	OUT	1	Unified Flow Key Buffer to Lookup and Update Engine Error. Asserted if a read request times out.
LUEnUFKBRdReq	IN	1	Lookup and Update Engine to Unified Flow Key Buffer Read Request.
LUEnUFKBWrStb	IN	1	Lookup and Update Engine to Unified Flow Key Buffer Write Strobe.

10.1.6.3 Lookup and Update Engine Interface

Signal	Dir	Width	Description
LUEDone	IN	1	Lookup and Update Engine Done. This input is used to tell the Unified Flow Key Buffer that the Lookup and Update Engine has finished with the current flow. The Unified Flow Key Buffer also uses this signal to increment it's internal pointer so that the next address from Lookup and Update Engine will point to the next flow buffer.
LUEHoldBuf	IN	1	Lookup and Update Engine Hold Buffer. This input is used to tell the Unified Flow Key Buffer that the Lookup and Update Engine is transferring processing of this buffer to the State Processor.

10.1.6.4 State Processor Interface

Signal	Dir	Width	Description
UFKBuSPData	OUT	*	Unified Flow Key Buffer to State Processor read Data bus. * Uses AN_UFKB_AWIDTH
SPrUFKBData	IN	*	State Processor to Unified Flow Key Buffer write Data bus. * Uses AN_UFKB_AWIDTH
SPrUFKBAdd	IN	*	State Processor to Unified Flow Key Buffer Address bus. * Uses AN_UFKB_AWIDTH
SPFlowKeyAv	OUT	1	State Processor Flow Key Available. This signal tells the State Processor that the Unified Flow Key Buffer module a flow key for it to process.
SPrUFKBWrStb	IN	1	State Processor to Unified Flow Key Buffer Write Strobe.
SPDone	IN	1	State Processor Done. This input is used to tell the Unified Flow Key Buffer that the State Processor has finished with the current flow. The Unified Flow Key Buffer also uses this signal to increment it's internal pointer so that the next address from State Processor will point to the next flow buffer.
SPHoldBuf	IN	1	State Processor Hold Buffer. This input is used to tell the Unified Flow Key Buffer that the State Processor is transferring processing of this buffer to the Flow Insertion and Deletion Engine.

10.1.6.5 Flow Insertion and Deletion Engine Interface

Signal	Dir	Width	Description
UFKBuFIDEData	OUT	*	Unified Flow Key Buffer to Flow Insertion and Deletion Engine read Data bus. * Uses AN_UFKB_AWIDTH
FIDEnUFKBAdd	IN	*	Flow Insertion and Deletion Engine to Unified Flow Key Buffer Address bus. * Uses AN_UFKB_AWIDTH
FIDEFlowKeyAv	OUT	1	Flow Insertion and Deletion Engine Flow Key Available. This signal tells the Flow Insertion and Deletion Engine that the Unified Flow Key Buffer module a flow key for it to process.

10.1.6.5 Flow Insertion and Deletion Engine Interface

Signal	Dir	Width	Description
FIDEDone	IN	1	Flow Insertion and Deletion Engine Done. This input is used to tell the Unified Flow Key Buffer that the Flow Insertion and Deletion Engine has finished with the current flow. The Unified Flow Key Buffer also uses this signal to increment it's internal pointer so that the next address from Flow Insertion and Deletion Engine will point to the next flow buffer.

10.1.7 Verilog Module

```

module UFKB(Reset_N, MCLK ,AnalyzerEn ,AnalyzerReady ,AnalyzerAbort
,ParserKeyDelim ,ParserDataAvail ,UFKBuLUEData ,LUEnUFKBData
,LUEnUFKBAdd ,FlowKeySt ,UFKBuLUERdy ,UFKBuLUEErr ,LUEnUFKBRdReq
,LUEnUFKBWrStb ,LUEDone ,LUEHoldBuf ,UFKBuSPData ,SPrUFKBData
,SPrUFKBAdd ,SPFlowKeyAv ,SPrUFKBWrStb ,SPDone ,SPHoldBuf ,UFKBuFIDEData
,FIDEnUFKBAdd ,FIDEFlowKeyAv ,FIDEDone);

```

```

// General Interface Interface
input Reset_N;
input MCLK;
input AnalyzerEn;
// Parser Interface
output AnalyzerReady;
output AnalyzerAbort;
input ParserKeyDelim;
input ParserDataAvail;
// Lookup and Update Engine Interface
output [`AN_UFKB_DWIDTH-1 : 0] UFKBuLUEData;
input [`AN_UFKB_DWIDTH-1 : 0] LUEEnUFKBData;
input [`AN_UFKB_AWIDTH-1 : 0] LUEEnUFKBAdd;
output FlowKeySt;
output UFKBuLUERdy;
output UFKBuLUEErr;
input LUEEnUFKBRdReq;
input LUEEnUFKBWrStb;
input LUEDone;
input LUEHoldBuf;
// State Processor Interface
output [`AN_UFKB_DWIDTH-1 : 0] UFKBuSPData;
input [`AN_UFKB_DWIDTH-1 : 0] SPrUFKBData;
input [`AN_UFKB_AWIDTH-1 : 0] SPrUFKBAdd;
output SPFlowKeyAv;
input SPrUFKBWrStb;
input SPSDone;
input SPHoldBuf;
// Flow Insertion and Deletion Engine
output [`AN_UFKB_DWIDTH-1 : 0] UFKBuFIDEData;
input [`AN_UFKB_AWIDTH-1 : 0] FIDEnUFKBAdd;

```

output FIDEFlowKeyAv;
input FIDEDone;

10.1.8 VHDL Component

10.2 Lookup and Update Engine - LUE

10.2.1 Symbol

10.2.2 Highlights

- Looks up flow entries
- Compares flow key from parser to flow entries
- Updates packet count and byte count tables
- 64 bit byte count adder with early out
- Checks flow state to see if processing by the state processor is required

10.2.3 Description

The Lookup and Update Engine begins processing as soon as a flow key arrives from the parser. The first transfer from the parser contains a hash value that is used as an offset into the flow entry database. The LUE checks the entry to see if it matches the flow key by comparing the unique identification for that flow. If there is a match, the LUE updates the counters for the flow entry. The LUE also check the entry's flow state to see if the flow key needs to be sent to the state processor.

The Lookup and Update Engine also outputs on a special data bus, two 16 bit values. One value is a word from the flow key that can be a packet identifier or any thing else the design wants. The other is the protocol identifier for the flow. This can be programmed to output this data on every packet or only for packets that the corresponding flow is in the IDENTIFIED state.

10.2.4 Implementation Information

10.2.5 File Names

Top: LUE.v(hd)

Uses: AnalyzerConstants.v(hd)

10.2.6 Pin Descriptions

10.2.6.1 General Interface Signals

Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
AnalyzerEn	IN	1	Analyzer Enable bit from the control register

10.2.6.2 Unified Flow Key Buffer Interface

Signal	Dir	Width	Description
UFKBuLUEData	IN	*	Unified Flow Key Buffer to Lookup and Update Engine read Data bus. * Uses AN_UFKB_DWIDTH

10.2.6.2 Unified Flow Key Buffer Interface			
Signal	Dir	Width	Description
LUEnUFKBData	OUT	*	Lookup and Update Engine to Unified Flow Key Buffer write Data bus. * Uses AN_UFKB_DWIDTH
LUEnUFKBAdd	OUT	*	Lookup and Update Engine to Unified Flow Key Buffer Address bus. * Uses AN_UFKB_AWIDTH
FlowKeySt	IN	1	Flow Key Start. This signal tells the Lookup and Update Engine that the Unified Flow Key Buffer module has placed the first word of a flow key buffer.
UFKBuLUERdy	IN	1	Unified Flow Key Buffer to Lookup and Update Engine Ready.
UFKBuLUEErr	IN	1	Unified Flow Key Buffer to Lookup and Update Engine Error. Asserted if a read request times out.
LUEnUFKBRdReq	OUT	1	Lookup and Update Engine to Unified Flow Key Buffer Read Request.
LUEnUFKBWrStb	OUT	1	Lookup and Update Engine to Unified Flow Key Buffer Write Strobe.
LUEDone	OUT	1	Lookup and Update Engine Done. This input is used to tell the Unified Flow Key Buffer that the Lookup and Update Engine has finished with the current flow. The Unified Flow Key Buffer also uses this signal to increment it's internal pointer so that the next address from Lookup and Update Engine will point to the next flow buffer.
LUEHoldBuf	OUT	1	Lookup and Update Engine Hold Buffer. This input is used to tell the Unified Flow Key Buffer that the Lookup and Update Engine is transferring processing of this buffer to the State Processor.

10.2.6.3 Cache Interface			
Signal	Dir	Width	Description
CaLUEReady	IN	1	Cache to Lookup Engine Ready. This signal tells the Lookup Engine that during a read, the data on the CaLUEData bus is valid and during a write that the Cache has latched the data on the LUEnCaData bus.
CaLUEData	IN	*	Cache to Lookup Engine Data bus. * Uses AN_CA_DWIDTH
LUEnCaData	OUT	*	Lookup Engine to Cache Data bus. * Uses AN_CA_DWIDTH
LUEAdd	OUT	*	Lookup Engine to Cache Address bus. * Uses AN_CA_AWIDTH
LUEMemReq	OUT	1	Lookup Engine Memory Request. If this signal is active, the address on the LUEAdd bus is valid.
LUEMemWr	OUT	1	Lookup Engine Memory Write. If this signal is active, the current transaction is a write..

10.2.6.4 Known Flow Interface			
Signal	Dir	Width	Description
PacketRef	OUT	*	Packet Reference number bus. This bus outputs the packet reference number copied from the UFKB. * Uses AN_FR_WIDTH
Protocol	OUT	*	Protocol bus. This bus outputs the highest level protocol the State Processor has determined the packet contains. * Uses AN_PRO_WIDTH
KnownFlowStb	OUT	1	Known Flow Strobe. When this signal is active, the data on the PacketRef and the Protocol busses are valid.

10.2.7 Verilog Module

```
module LUE(Reset_N, MCLK ,AnalyzerEn ,UFKBuLUEData ,LUEnUFKBData
,LUEnUFKBAdd ,FlowKeySt ,UFKBuLUERdy ,UFKBuLUEErr ,LUEnUFKBRdReq
,LUEnUFKBWrStb ,LUEDone ,LUEHoldBuf ,CaLUEData ,LUEnCaData ,LUEAdd
,LUEMemReq ,LUEMemWr);
```

```
// General Interface Interface
input Reset_N;
input MCLK;
input AnalyzerEn;
// Unified Flow Key Buffer Interface
input [`AN_UFKB_DWIDTH-1 : 0] UFKBuLUEData;
output [`AN_UFKB_DWIDTH-1 : 0] LUEuUFKBData;
output [`AN_UFKB_AWIDTH-1 : 0] LUEuUFKBAdd;
input FlowKeySt;
input UFKBuLUERdy;
input UFKBuLUEErr;
output LUEuUFKBRdReq;
output LUEuUFKBWrStb;
output LUEDone;
output LUEHoldBuf;
// Cache Interface
input CaLUEReady;
input [`AN_CA_DWIDTH-1 : 0] CaLUEData;
output [`AN_CA_DWIDTH-1 : 0] LUEuCaData;
output [`AN_CA_AWIDTH-1 : 0] LUEAdd;
output LUEMemReq;
output LUEMemWr;
// Known Flow Interface
output [`AN_FR_WIDTH-1 : 0] PacketRef;
output [`AN_PRO_WIDTH-1 : 0] Protocol;
output KnownFlowStb;
```

10.2.8 VHDL Component

10.3 Analyzer CPU Interface and Control - ACIC

10.3.1 Symbol

10.3.2 Description

The Analyzer CPU Interface Control module controls the communication between the external CPU and the Analyzer. The ACIC contains MUX's for the CPU read back path. It also contains the control register for the Analyzer.

10.3.3 File Names

Top: ACIC.v(hd)

Uses: AnalyzerConstants.v(hd)

10.3.4 Pin Descriptions

10.3.4.1 General Interface Signals

Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
AnalyzerEn	OUT	1	Analyzer Enable bit from the control register

10.3.4.2 Host Interface Signals

Signal	Dir	Width	Description
AnalyzerSel_N	IN	1	Host interface Analyzer Select - active low. AnalyzerSel_N is sampled on the rising edge of MCLK . If it is active, it signifies that the external host is attempting to access the analyzer.
HostWrite	IN	1	Write. Write is sampled on the rising edge of MCLK . This signal is only valid when AnalyzerSel_N is active. If this signal is active, the host is attempting to write to the analyzer. Inactive this signal signifies a read from the analyzer. It should also be used to control the direction of the host data bus if it is bidirectional.
HostBlast_N	IN	1	Burst Last - active low. HostBlast_N is sampled on the rising edge of MCLK . HostBlast_N tells the analyzer that the current transfer is the last transfer in this burst.
HostWait_N	IN	1	Wait - active low. HostWait_N is sampled on the rising edge of MCLK . The host asserts HostWait_N when it wishes to slow transfers between itself and the analyzer. This could also be used by additional interface logic to slow transfers so it can multiplex the bus down to a smaller size without additional FIFOs. If wait is active, HostReady_N is blocked.

10.3.4.2 Host Interface Signals

Signal	Dir	Width	Description
AnaHostReady_N	OUT	1	Analyzer to Host Ready – active low. AnaHostReady_N should be sampled on the rising edge of MCLK . The analyzer returns AnaHostReady_N when the current cycle is completed. For a write operation, AnaHostReady_N means that the HostDataIn bus has been latched. For a read operation AnaHostReady_N means that the requested data is on the HostDataOut bus and is valid. AnaHostReady_N is blocked by HostWait_N .
HostAddress	IN	*	Host Address bus. HostAddress is sampled on the rising edge of MCLK if AnalyzerSel_N is active. This bus defines the first address in this burst to access in the 32 Megabyte address space of the analyzer. See Section x.x.x for the Address Utilization Map. * Uses AN_HOST_AWIDTH
HostByteEn_N	IN	*	Host Byte Enable bus – Active low. HostWait_N is sampled on the rising edge of MCLK . * Uses AN_HOST_BEWIDTH
HostDataIn	IN	*	Host Data Input bus. HostDataIn is sampled on the rising edge of MCLK if HostWrite is active and HostWait_N is inactive. * Uses AN_HOST_DWIDTH
AnaHostDataOut	OUT	*	Analyzer Host Data Output bus. AnaHostDataOut should be sampled on the rising edge of MCLK . Data on this bus is valid during a read cycle when AnaHostReady_N is active. * Uses AN_HOST_DWIDTH

10.3.4.3 Cache Interface

Signal	Dir	Width	Description
CaACICReady	IN	1	Cache to Analyzer CPU Interface Control Ready. This signal tells the Analyzer CPU Interface Control that during a read, the data on the CaACICData bus is valid and during a write that the Cache has latched the data on the ACICnCaData bus.
CaACICData	IN	*	Cache to Analyzer CPU Interface Control Data bus. * Uses AN_CA_DWIDTH
ACICoCaData	OUT	*	Analyzer CPU Interface Control to Cache Data bus. * Uses AN_CA_DWIDTH
ACICAdd	OUT	*	Analyzer CPU Interface Control to Cache Address bus. * Uses AN_CA_AWIDTH
ACICMemReq	OUT	1	Analyzer CPU Interface Control Memory Request. If this signal is active, the address on the ACICAdd bus is valid.
ACICMemWr	OUT	1	Analyzer CPU Interface Control Memory Write. If this signal is active, the current transaction is a write..

10.3.4.4 State Processor Instruction Database Interface			
Signal	Dir	Width	Description
ACICoSPIDWr	OUT	1	Analyzer CPU Interface Control to State Processor Instruction Database Write Strobe
ACICoSPIDAdd	OUT	*	Analyzer CPU Interface Control to State Processor Instruction Database Address bus * uses AN_SPID_AWIDTH
SPIDData	IN	*	State Processor Instruction Database Data bus * uses AN_SPID_DWIDTH
ACICoSPIDData	OUT	*	Analyzer CPU Interface Control to State Processor Instruction Database Data bus * uses AN_SPID_DWIDTH

10.3.5 Verilog Module

```
module ACIC(Reset_N, MCLK ,AnalyzerEn ,AnalyzerSel_N ,HostWrite
,HostBlast_N ,HostWait_N ,AnaHostReady_N ,HostAddress ,HostByteEn_N
,HostDataIn ,AnaHostDataOut ,CaACICReady ,CaACICData ,ACICoCaData
,ACICAdd , ACICMemReq ,ACICMemWr ,ACICoSPIDWr ,ACICoSPIDAdd ,SPIDData
,ACICoSPIDData);
```

```
// General Interface Interface
input Reset_N;
input MCLK;
output AnalyzerEn;
// Host Interface
input AnalyzerSel_N;
input HostWrite;
input HostBlast_N;
input HostWait_N;
output AnaHostReady_N;
input [`AN_HOST_AWIDTH-1 : 0] HostAddress;
input [`AN_HOST_BEWIDTH-1 : 0] HostByteEn_N;
input [`AN_HOST_DWIDTH-1 : 0] HostDataIn;
output [`AN_HOST_DWIDTH-1 : 0] AnaHostDataOut;
// Cache Interface
input CaACICReady;
input [`AN_CA_DWIDTH-1 : 0] CaACICData;
output [`AN_CA_DWIDTH-1 : 0] ACICoCaData;
output [`AN_CA_AWIDTH-1 : 0] ACICAdd;
output ACICMemReq;
output ACICMemWr;
// State Processor Instruction Database Interface
output ACICoSPIDWr;
output [`AN_SPID_AWIDTH-1 : 0] ACICoSPIDAdd;
input [`AN_SPID_DWIDTH-1 : 0] SPIDData;
output [`AN_SPID_DWIDTH-1 : 0] ACICoSPIDData;
```

10.3.6 VHDL Component

10.4 Flow Insertion and Deletion Engine - FIDE

10.4.1 Symbol

10.4.2 Highlights

- Maintains flow entry database
- Deletes and inserts flows based on a LRU algorithm
- Builds flows from flow key and State Processor instructions

10.4.3 Description

The Flow Insertion and Deletion Engine maintains the flow entry database. Flows are grouped into buckets by hash value. When a new flow needs to be inserted first the FIDE sees which of the entries in the corresponding bucket is the oldest. It then builds the flow entry from the flow key and State Processor instructions. Finally it places the entry in the database.

10.4.4 Implementation Information

10.4.5 File Names

Top: FIDE.v(hd)

Uses: AnalyzerConstants.v(hd)

10.4.6 Pin Descriptions

10.4.6.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
AnalyzerEn	IN	1	Analyzer Enable bit from the control register

10.4.6.2 Unified Flow Key Buffer Interface			
Signal	Dir	Width	Description
UFKBuFIDEData	IN	*	Unified Flow Key Buffer to Flow Insertion and Deletion Engine read Data bus. * Uses AN_UFKB_AWIDTH
FIDEnUFKBAdd	OUT	*	Flow Insertion and Deletion Engine to Unified Flow Key Buffer Address bus. * Uses AN_UFKB_AWIDTH
FIDEFlowKeyAv	IN	1	Flow Insertion and Deletion Engine Flow Key Available. This signal tells the Flow Insertion and Deletion Engine that the Unified Flow Key Buffer module a flow key for it to process.

10.4.6.2 Unified Flow Key Buffer Interface

Signal	Dir	Width	Description
FIDEDone	OUT	1	Flow Insertion and Deletion Engine Done. This input is used to tell the Unified Flow Key Buffer that the Flow Insertion and Deletion Engine has finished with the current flow. The Unified Flow Key Buffer also uses this signal to increment it's internal pointer so that the next address from Flow Insertion and Deletion Engine will point to the next flow buffer.

10.4.6.3 Cache Interface

Signal	Dir	Width	Description
CaFIDEReady	IN	1	Cache to Flow Insertion and Deletion Engine Ready. This signal tells the Flow Insertion and Deletion Engine that during a read, the data on the CaFIDEData bus is valid and during a write that the Cache has latched the data on the FIDEnCaData bus.
CaFIDEData	IN	*	Cache to Flow Insertion and Deletion Engine Data bus. * Uses AN_CA_DWIDTH
FIDEnCaData	OUT	*	Flow Insertion and Deletion Engine to Cache Data bus. * Uses AN_CA_DWIDTH
FIDEAdd	OUT	*	Flow Insertion and Deletion Engine to Cache Address bus. * Uses AN_CA_AWIDTH
FIDEMemReq	OUT	1	Flow Insertion and Deletion Engine Memory Request. If this signal is active, the address on the FIDEAdd bus is valid.
FIDEMemWr	OUT	1	Flow Insertion and Deletion Engine Memory Write. If this signal is active, the current transaction is a write..

10.4.7 Verilog Module

```
module FIDE(Reset_N, MCLK ,AnalyzerEn ,UFKBuFIDEData ,FIDEnUFKBAdd
,FIDEFlowKeyAv ,FIDEDone ,CaFIDEData ,FIDEnCaData ,FIDEAdd ,FIDEMemReq
,FIDEMemWr);
```

```
// General Interface Interface
input Reset_N;
input MCLK;
input AnalyzerEn;
// Unified Flow Key Buffer Interface
input [`AN_UFKB_DWIDTH-1 : 0] UFKBuFIDEData;
output [`AN_UFKB_AWIDTH-1 : 0] FIDEnUFKBAdd;
input FIDEFlowKeyAv;
output FIDEDone;
// Cache Interface
input CaFIDEReady;
input [`AN_CA_DWIDTH-1 : 0] CaFIDEData;
output [`AN_CA_DWIDTH-1 : 0] FIDEnCaData;
```

```
output [`AN_CA_AWIDTH-1 : 0] FIDEAdd;  
output FIDEMemReq;  
output FIDEMemWr;
```

10.4.8 VHDL Component



10.5 State Processor Instruction Database - SPID

10.5.1 Symbol

10.5.2 Highlights

- Scaleable implementation
- Wraps either RAM or ROM instantiation or can be synthesized latches

10.5.3 Description

The State Processor Instruction Database module is a wrapper for the storage medium used to hold the State Processor Instruction database. Only the CPU can write this memory. The CPU interface is active if **AnalyzerEn** is active.

10.5.4 Implementation Information

The module can be synthesized or a RAM or ROM cell can be instantiated into the wrapper.

10.5.5 File Names

Top: SPID.v(hd)

Uses: AnalyzerConstants.v(hd)

10.5.6 Pin Descriptions

10.5.6.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
AnalyzerEn	IN	1	Analyzer Enable bit from the control register

10.5.6.2 Analyzer CPU Interface Control Interface			
Signal	Dir	Width	Description
ACICoSPIDWr	IN	1	Analyzer CPU Interface Control to State Processor Instruction Database Write Strobe
ACICoSPIDAdd	IN	*	Analyzer CPU Interface Control to State Processor Instruction Database Address bus * uses AN_SPID_AWIDTH
SPIDData	OUT	*	State Processor Instruction Database Data bus * uses AN_SPID_DWIDTH
ACICoSPIDData	IN	*	Analyzer CPU Interface Control to State Processor Instruction Database Data bus * uses AN_SPID_DWIDTH

10.5.6.3 State Processor Interface

Signal	Dir	Width	Description
SPrSPIDAdd	IN	*	State Processor to State Processor Instruction Database Address bus * uses AN_SPID_AWIDTH

10.5.7 Verilog Module

```
module SPID(Reset_N, MCLK ,AnalyzerEn ,ACICoSPIDWr ,ACICoSPIDAdd  
,SPIDData ,ACICoSPIDData ,SPrSPIDAdd);
```

```
// General Interface Interface  
input Reset_N;  
input MCLK;  
input AnalyzerEn;  
// Analyzer CPU Interface Control Interface  
input ACICoSPIDWr;  
input [`AN_SPID_AWIDTH-1 : 0] ACICoSPIDAdd;  
output [`AN_SPID_DWIDTH-1 : 0] SPIDData;  
input [`AN_SPID_DWIDTH-1 : 0] ACICoSPIDData;  
// State Processor Interface  
input [`AN_SPID_AWIDTH-1 : 0] SPrSPIDAdd;
```

10.5.8 VHDL Component

10.6 Unified Memory Controller - UMC

10.6.1 Symbol

10.6.2 Highlights

- Supports Both SDRAM and SGRAM
- Maintains RAM refresh

10.6.3 Description

The Unified Memory Controller module controls the caches' access to the flow database contained in external RAM. Synchronous DRAM is controlled through a series of instructions feed to the RAM through the control pins. Synchronous DRAM requires at startup a specific series of commands for initialization. The Unified Memory Controller handles both processes thorough a state machine. Since the nature of the flow database requires random access, there is little use in attempting to keep multiple banks open. Auto-refresh is continuous when memory is not being accessed by the cache.

10.6.4 Implementation Information

The Unified Memory Controller module is implemented as a Moore type finite state machine. Each of the outputs of the state machine are registered to assure maximum setup time for the external device.

10.6.5 File Names

Top: UMC.v(hd)

Uses: AnalyzerConstants.v(hd)

10.6.6 Pin Descriptions

10.6.6.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
AnalyzerEn	IN	1	Analyzer Enable bit from the control register

10.6.6.2 Memory Interface			
Signal	Dir	Width	Description
MemClkIn	IN	1	Memory clock in. This signal is used to generate the memory interface timing.
MemRAS_N	OUT	*	Memory Row Address Strobe bus – active low. * uses AN_MEM_RASWIDTH
MemCAS_N	OUT	*	Memory Column Address Strobe bus– active low. * uses AN_MEM_CASWIDTH
MemClkEn	OUT	1	Memory Clock Enable. Some memories require this signal to be disabled for a certain amount of time after reset.

10.6.6.2 Memory Interface

Signal	Dir	Width	Description
MemClkOut	OUT	1	Memory Clock Out. This signal is used by synchronous memory for all operations. MemClkIn is buffered and sent out on this pin. This helps reduce skew between this clock and the other signals.
MemWR_N	OUT	1	Memory Write – active low.
MemBA	OUT	1	Memory Bank Address. Used by multi-bank memory to select the bank the current operation is to operate on.
MemDSF	OUT	1	Memory Special Function select.
MemByteEn_N	OUT	*	Memory Byte Enable bus– active low. * uses AN_MEM_BEWIDTH
MemAddress	OUT	*	Memory Address bus. * uses AN_MEM_AWIDTH
MemDataIn	IN	*	Memory Data Input bus. * uses AN_MEM_DWIDTH
MemDataOut	OUT	*	Memory Data Output bus. * uses AN_MEM_DWIDTH
MemDirRead	OUT	1	Memory Data bus Direction is Read. This signal is used to control the tri-state enable on the bidirectional memory data bus. If MemDirRead is active data is coming into the analyzer from the memory. If it is inactive the analyzer is driving data out to the memory.

10.6.6.3 Cache Interface

Signal	Dir	Width	Description
UMCoCaReady	IN	1	Unified Memory Controller to Cache Ready. This signal tells the Cache that during a read, the data on the UMCoCaData bus is valid and during a write that the Unified Memory Controller has latched the data on the CaUMCData bus.
UMCoCaData	IN	*	Unified Memory Controller to Cache Data bus. * Uses AN_CA_DWIDTH
CaUMCData	OUT	*	Cache to Unified Memory Controller Data bus. * Uses AN_CA_DWIDTH
CaUMCAdd	OUT	*	Cache to Unified Memory Controller Address bus. * Uses AN_CA_AWIDTH
CaMemReq	OUT	1	Cache Memory Request. If this signal is active, the address on the CaUMCAdd bus is valid.
CaMemWr	OUT	1	Cache Memory Write. If this signal is active, the current transaction is a write..

10.6.7 Verilog Module

```
module UMC(Reset_N, MCLK ,AnalyzerEn ,MemClkIn ,MemRAS_N ,MemCAS_N
,MemClkEn ,MemClkOut ,MemWR_N ,MemBA ,MemDSF ,MemByteEn_N ,MemAddress
```

,MemDataIn ,MemDataOut ,MemDirRead ,UMCoCaReady ,UMCoCaData ,CaUMCData
,CaUMCAdd ,CaMemReq ,CaMemWr);

```
// General Interface Interface
input Reset_N;
input MCLK;
input AnalyzerEn;
// Memory Interface
input MemClkIn;
output [`AN_MEM_RASWIDTH-1 : 0] MemRAS_N;
output [`AN_MEM_CASWIDTH-1 : 0] MemCAS_N;
output MemClkEn;
output MemClkOut;
output MemWR_N;
output MemBA;
output MemDSF;
output [`AN_MEM_BEWIDTH-1 : 0] MemByteEn_N;
output [`AN_MEM_AWIDTH-1 : 0] MemAddress;
input [`AN_MEM_DWIDTH-1 : 0] MemDataIn;
output [`AN_MEM_DWIDTH-1 : 0] MemDataOut;
output MemDirRead;
// Cache Interface
input UMCaCaReady;
input [`AN_CA_DWIDTH-1 : 0] UMCaCaData;
output [`AN_CA_DWIDTH-1 : 0] CaUMCData;
output [`AN_CA_AWIDTH-1 : 0] CaUMCAdd;
output CaMemReq;
output CaMemWr;
```

10.6.8 VHDL Component

10.7 Cache

10.7.1 Symbol

10.7.2 Highlights

- Fully associative
- True least recently used cache updating
- Simultaneous one write and two reads.

10.7.3 Description

The Cache module contains a fully associative, true LRU cache memory. Full associativity is achieved through the use of a content addressable memory (CAM). The need for a fully associative cache arises from the fact that the hash uses to generate the initial look up into the flow entry database spreads the entries pseudo randomly throughout the memory. Each hash value corresponds to a bucket containing N flow entries. N is set by the designer (see section xxx).

The Cache can service two read transfers at one time. If there are more than two read requests active at one time the Cache services them in the order shown in section xxx.

The CAM contains the hash value associated with the corresponding bucket in the cache memory. When there is a cache hit, the CAM produces the most significant bits of the address in cache memory where the bucket is stored. The cache then accesses the cache memory at the address indicated concatenating the lower address bits provided by the requesting module. The cache then remembers that the requesting module had a cache hit and the memory location returned. This allows a cache lookup for a requesting module to occur only once per request. When the requesting module requires a different bucket, it drops then again raises its request and another CAM cycle is initiated.

The least recently used algorithm requires the CAM to also be a stack. When there is a cache hit the CAM location that produced the hit is put on the top of the stack. The other locations above the hit location are shifted down to fill in the gap. If there is a miss, the bottom location is read to determine the address in the cache memory to put the new bucket. All the locations shifted down as normally. Finally the new hash value and cache memory address are put at the top of the stack.

10.7.3.1 Priority

The Cache processes requests from the attached modules in the following order:

- 1 - LRU dirty write back. The Cache writes back the least recently used bucket if it is dirty so that there will always be a space for the fetching of cache misses.
- 2 - Lookup and Update Engine.
- 3 - State Processor.
- 4 - Flow Insertion and Deletion Engine.
- 5 - Analyzer CPU Interface and Control
- 6 - Dirty write back from LRU -1 to MRU. When there is nothing else pending the Cache writes dirty entries back to memory.

10.7.4 Implementation Information

10.7.5 File Names

Top: Cache.v(hd)

Uses: AnalyzerConstants.v(hd)

10.7.6 Pin Descriptions

10.7.6.1 General Interface Signals			
Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
AnalyzerEn	IN	1	Analyzer Enable bit from the control register

10.7.6.2 Unified Memory Controller Interface			
Signal	Dir	Width	Description
UMCoCaReady	OUT	1	Unified Memory Controller to Cache Ready. This signal tells the Cache that during a read, the data on the UMCoCaData bus is valid and during a write that the Unified Memory Controller has latched the data on the CaUMCData bus.
UMCoCaData	OUT	*	Unified Memory Controller to Cache Data bus. * Uses AN_CA_DWIDTH
CaUMCData	IN	*	Cache to Unified Memory Controller Data bus. * Uses AN_CA_DWIDTH
CaUMCAdd	IN	*	Cache to Unified Memory Controller Address bus. * Uses AN_CA_AWIDTH
CaMemReq	IN	1	Cache Memory Request. If this signal is active, the address on the CaUMCAdd bus is valid.
CaMemWr	IN	1	Cache Memory Write. If this signal is active, the current transaction is a write..

10.7.6.3 Flow Insertion and Deletion Engine Interface			
Signal	Dir	Width	Description
CaFIDEReady	OUT	1	Cache to Flow Insertion and Deletion Engine Ready. This signal tells the Flow Insertion and Deletion Engine that during a read, the data on the CaFIDEData bus is valid and during a write that the Cache has latched the data on the FIDEnCaData bus.
CaFIDEData	OUT	*	Cache to Flow Insertion and Deletion Engine Data bus. * Uses AN_CA_DWIDTH
FIDEnCaData	IN	*	Flow Insertion and Deletion Engine to Cache Data bus. * Uses AN_CA_DWIDTH
FIDEAdd	IN	*	Flow Insertion and Deletion Engine to Cache Address bus. * Uses AN_CA_AWIDTH

10.7.6.3 Flow Insertion and Deletion Engine Interface

Signal	Dir	Width	Description
FIDEMemReq	IN	1	Flow Insertion and Deletion Engine Memory Request. If this signal is active, the address on the FIDEAdd bus is valid.
FIDEMemWr	IN	1	Flow Insertion and Deletion Engine Memory Write. If this signal is active, the current transaction is a write..

10.7.6.4 Analyzer CPU Interface Control Interface

Signal	Dir	Width	Description
CaACICReady	OUT	1	Cache to Analyzer CPU Interface Control Ready. This signal tells the Analyzer CPU Interface Control that during a read, the data on the CaACICData bus is valid and during a write that the Cache has latched the data on the ACICnCaData bus.
CaACICData	OUT	*	Cache to Analyzer CPU Interface Control Data bus. * Uses AN_CA_DWIDTH
ACICoCaData	IN	*	Analyzer CPU Interface Control to Cache Data bus. * Uses AN_CA_DWIDTH
ACICAdd	IN	*	Analyzer CPU Interface Control to Cache Address bus. * Uses AN_CA_AWIDTH
ACICMemReq	IN	1	Analyzer CPU Interface Control Memory Request. If this signal is active, the address on the ACICAdd bus is valid.
ACICMemWr	IN	1	Analyzer CPU Interface Control Memory Write. If this signal is active, the current transaction is a write..

10.7.6.5 Lookup Engine Interface

Signal	Dir	Width	Description
CaLUEReady	OUT	1	Cache to Lookup Engine Ready. This signal tells the Lookup Engine that during a read, the data on the CaLUEDData bus is valid and during a write that the Cache has latched the data on the LUEnCaData bus.
CaLUEDData	OUT	*	Cache to Lookup Engine Data bus. * Uses AN_CA_DWIDTH
LUEnCaData	IN	*	Lookup Engine to Cache Data bus. * Uses AN_CA_DWIDTH
LUEAdd	IN	*	Lookup Engine to Cache Address bus. * Uses AN_CA_AWIDTH
LUEMemReq	IN	1	Lookup Engine Memory Request. If this signal is active, the address on the LUEAdd bus is valid.
LUEMemWr	IN	1	Lookup Engine Memory Write. If this signal is active, the current transaction is a write..



10.7.6.6 State Processor Interface			
Signal	Dir	Width	Description
CaSPReady	OUT	1	Cache to State Processor Ready. This signal tells the Lookup Engine that during a read, the data on the CaSPData bus is valid and during a write that the Cache has latched the data on the SPnCaData bus.
CaSPData	OUT	*	Cache to State Processor Data bus. * Uses AN_CA_DWIDTH
SPnCaData	IN	*	State Processor to Cache Data bus. * Uses AN_CA_DWIDTH
SPAdd	IN	*	State Processor to Cache Address bus. * Uses AN_CA_AWIDTH
SPMemReq	IN	1	State Processor Memory Request. If this signal is active, the address on the SPAdd bus is valid.
SPMemWr	IN	1	State Processor Memory Write. If this signal is active, the current transaction is a write..

10.7.7 Verilog Module

```
module Cache(Reset_N, MCLK ,AnalyzerEn ,UMCoCaReady ,UMCoCaData
,CaUMCData ,CaUMCAdd ,CaMemReq ,CaMemWr ,CaFIDEReady ,CaFIDEData
,FIDEnCaData ,FIDEAdd ,FIDEMemReq ,FIDEMemWr ,CaACICReady ,CaACICData
,ACICoCaData ,ACICAdd ,ACICMemReq ,ACICMemWr ,CaLUEReady ,CaLUEData
,LUEnCaData ,LUEAdd ,LUEMemReq ,LUEMemWr ,CaSPReady ,CaSPData ,SPnCaData
,SPAdd ,SPMemReq ,SPMemWr);
```

```
// General Interface Interface
input Reset_N;
input MCLK;
input AnalyzerEn;
// Unified Memory Controller Interface
output UMCoCaReady;
output [`AN_CA_DWIDTH-1 : 0] UMCoCaData;
input [`AN_CA_DWIDTH-1 : 0] CaUMCData;
input [`AN_CA_AWIDTH-1 : 0] CaUMCAdd;
input CaMemReq;
input CaMemWr;
// Flow Insertion and Deletion Engine Interface
output CaFIDEReady;
output [`AN_CA_DWIDTH-1 : 0] CaFIDEData;
input [`AN_CA_DWIDTH-1 : 0] FIDEnCaData;
input [`AN_CA_AWIDTH-1 : 0] FIDEAdd;
input FIDEMemReq;
input FIDEMemWr;
// Analyzer CPU Interface Control Interface
output CaACICReady;
output [`AN_CA_DWIDTH-1 : 0] CaACICData;
input [`AN_CA_DWIDTH-1 : 0] ACICoCaData;
input [`AN_CA_AWIDTH-1 : 0] ACICAdd;
input ACICMemReq;
```



```
input ACICMemWr;  
// Lookup Engine Interface  
output CaLUEReady;  
output [`AN_CA_DWIDTH-1 : 0] CaLUEData;  
input [`AN_CA_DWIDTH-1 : 0] LUEnCaData;  
input [`AN_CA_AWIDTH-1 : 0] LUEAdd;  
input LUEMemReq;  
input LUEMemWr;  
// State Processor Interface  
output CaSPReady;  
output [`AN_CA_DWIDTH-1 : 0] CaSPData;  
input [`AN_CA_DWIDTH-1 : 0] SPnCaData;  
input [`AN_CA_AWIDTH-1 : 0] SPAdd;  
input SPMemReq;  
input SPMemWr;
```

10.7.8 VHDL Component

10.8 State Processor - SP

10.8.1 Symbol

10.8.2 Highlights

- Flexible Rule-based Traffic Classification
- State-based Tracking of Traffic
- **Multiple** Packets for Layer Processing
- Programmable Rules/State Processor
- Selectable Protocols in Flows
- Future Protocols Support

10.8.3 Description

The State Processor module analyzes both new and existing flows in order to classify them by application. It does this by proceeding from state to state based on rules defined by the engineer. A rule is a test followed by the next state to proceed to if the test is true. The State Processor goes through each rule until the test is true or there are no more tests to perform. The State Processor starts the process by using the last protocol recognized by the Parser as an offset into a jump table. The jump table takes us to the instructions to use for that protocol. Most instructions test something in the Unified Flow Key Buffer or the flow entry if it exists. The State Processor may have to test bits, do comparisons, add or subtract to perform the test.

10.8.4 Architecture

The State Processor contains several sub-blocks:

10.8.4.1 Scratch Pad Registers

The State Processor contains four scratch pad registers. These registers are the source and/or the destination for all instructions. It is implemented as a register file with one write and two read ports.

10.8.4.2 Instruction Pointer and Stack

The Instruction Pointer is used to point to the State Processor Instruction Database address that the State Processor is executing. The Instruction Pointer is initialized with the last protocol recognized by the Parser. This first instruction is a jump to the subroutine where the protocol is decoded. The State Processor supports calls so the Instruction Pointer block contains a two level stack. A one bit stack pointer points to the top of the stack that the Instruction Pointer is pushed to or popped from.

10.8.4.3 Flag Register

The Flag Register contains several bits used for conditional branching.

10.8.4.3.1 Flag Register Word Definition	
Bit	Description

10.8.4.3.1 Flag Register Word Definition

Bit	Description

10.8.4.4 Compare Block

The Compare Block compares two operands by exclusive-oring them together. The Compare Mask Register is contained in this block. If a bit is set in the Compare Mask Register, that bit is ignored in the compare operation.

10.8.4.5 Flow Key Pointer

The Flow Key Pointer provides the address that the State Processor is accessing in the Unified Flow Key Buffer. The Flow Key Pointer can perform both direct and indirect addressing. Indirect addressing is used to offset into a protocols' header.

10.8.4.6 Flow Entry Pointer

The Flow Entry Pointer provides the address that the State Processor is accessing in the Flow Entry in the Cache. If the flow entry exists, the upper address bits come from the hash used to lookup the bucket in the Flow database. The middle bits come from the bucket entry found. The lower bits come from the offset the State Processor is using.

10.8.5 Instruction Definitions

The following sections describe the instructions available in the State Processor. It should be noted that no assembler is provided for the State Processor. This is because the engineer need not write code for this processor. The MeterFlow Compiler writes the database entered into the State Processor Instruction Database from the protocols defined in the Protocol List.

10.8.5.1 Jump

This instruction causes the Instruction Pointer to be loaded with the address in the JumpAddress field of the State Processor Instruction Database. This instruction is always conditional. Whether the branch is taken or not depends on the on the ConditionCode field in the instruction and the state of the flag register.

10.8.5.2 Call

This instruction causes the Instruction Pointer to be loaded with the address in the JumpAddress field of the State Processor Instruction Database. At the same time the current address in the Instruction Pointer is pushed onto the stack. This instruction is always conditional. Whether the call is taken made or not depends on the on the ConditionCode field in the instruction and the state of the flag register.

10.8.5.3 Return

This instruction causes the Instruction Pointer to be loaded with the address at the top of the stack. This instruction is always conditional. Whether the return is executed or not depends on the on the ConditionCode field in the instruction and the state of the flag register.

10.8.5.4 Copy

The Copy instruction moves data from:

- Flow Key to Scratch Pad Register
- Cache to Scratch Pad Register
- ImmediateData to Scratch Pad Register
- Scratch Pad Register to Flow Key
- Scratch Pad Register to Cache
- Scratch Pad Register to Compare Mask Register

The external address can be either a direct or indirect access.

10.8.5.5 Compare

This instruction compares two operands . The operands must be either from a Scratch Pad Register or an immediate value from the instruction's ImmediateData field. The Compare Mask Register is used to set bit to don't care.

10.8.5.6 Instruction Word Definition

Bit	Description

10.8.6 Implementation Information

10.8.7 File Names

Top: SP.v(hd)

Uses: AnalyzerConstants.v(hd)

10.8.8 Pin Descriptions

10.8.8.1 General Interface Signals

Signal	Dir	Width	Description
Reset_N	IN	1	Reset - active low.
MCLK	IN	1	Module Clock.
AnalyzerEn	IN	1	Analyzer Enable bit from the control register

10.8.8.2 Unified Flow Key Buffer Interface

Signal	Dir	Width	Description
--------	-----	-------	-------------

10.8.8.2 Unified Flow Key Buffer Interface

Signal	Dir	Width	Description
UFKBuSPData	IN	*	Unified Flow Key Buffer to State Processor read Data bus. * Uses AN_UFKB_AWIDTH
SPrUFKBData	OUT	*	State Processor to Unified Flow Key Buffer write Data bus. * Uses AN_UFKB_AWIDTH
SPrUFKBAdd	OUT	*	State Processor to Unified Flow Key Buffer Address bus. * Uses AN_UFKB_AWIDTH
SPFlowKeyAv	IN	1	State Processor Flow Key Available. This signal tells the State Processor that the Unified Flow Key Buffer module a flow key for it to process.
SPrUFKBWrStb	OUT	1	State Processor to Unified Flow Key Buffer Write Strobe.
SPDone	OUT	1	State Processor Done. This input is used to tell the Unified Flow Key Buffer that the State Processor has finished with the current flow. The Unified Flow Key Buffer also uses this signal to increment it's internal pointer so that the next address from State Processor will point to the next flow buffer.
SPHoldBuf	OUT	1	State Processor Hold Buffer. This input is used to tell the Unified Flow Key Buffer that the State Processor is transferring processing of this buffer to the Flow Insertion and Deletion Engine.

10.8.8.3 Cache Interface

Signal	Dir	Width	Description
CaSPReady	IN	1	Cache to State Processor Ready. This signal tells the Lookup Engine that during a read, the data on the CaSPData bus is valid and during a write that the Cache has latched the data on the SPnCaData bus.
CaSPData	IN	*	Cache to State Processor Data bus. * Uses AN_CA_DWIDTH
SPrCaData	OUT	*	State Processor to Cache Data bus. * Uses AN_CA_DWIDTH
SPAdd	OUT	*	State Processor to Cache Address bus. * Uses AN_CA_AWIDTH
SPMemReq	OUT	1	State Processor Memory Request. If this signal is active, the address on the SPAdd bus is valid.
SPMemWr	OUT	1	State Processor Memory Write. If this signal is active, the current transaction is a write..

10.8.8.4 State Processor Interface

Signal	Dir	Width	Description
SPIDData	IN	*	State Processor to State Processor Instruction Database Data bus * uses AN_SPID_DWIDTH
SPrSPIDAdd	OUT	*	State Processor to State Processor Instruction Database Address bus * uses AN_SPID_AWIDTH

10.8.9 Verilog Module

```
module SP(Reset_N, MCLK ,AnalyzerEn ,UFKBuSPData ,SPrUFKBData
,SPrUFKBAdd ,SPFlowKeyAv ,SPnUFKBWrStb ,SPDone ,SPHoldBuf ,CaSPData
,SPrCaData ,SPAdd ,SPMemReq ,SPMemWr);

// General Interface Interface
input Reset_N;
input MCLK;
input AnalyzerEn;
// Unified Flow Key Buffer Interface
input [`AN_UFKB_DWIDTH-1 : 0] UFKBuSPData;
output [`AN_UFKB_DWIDTH-1 : 0] SPrUFKBData;
output [`AN_UFKB_AWIDTH-1 : 0] SPrUFKBAdd;
input SPFlowKeyAv;
output SPrUFKBWrStb;
output SPDone;
output SPHoldBuf;
// Cache Interface
input CaSPReady;
input [`AN_CA_DWIDTH-1 : 0] CaSPData;
output [`AN_CA_DWIDTH-1 : 0] SPrCaData;
output [`AN_CA_AWIDTH-1 : 0] SPAdd;
output SPMemReq;
output SPMemWr;
// State Processor Instruction Database
input [`AN_SPID_DWIDTH-1 : 0] SPIDData;
output [`AN_SPID_AWIDTH-1 : 0] SPrSPIDAdd;
```

10.8.10 VHDL Component

11 Appendix A - Multi-Packet State Processing

11.1 Overview

The MeterFlow Accelerator system is composed of four major subsystems. Each system interacts with the others by passing specific information and identification to parse, extract, generate flows and analyze single or multiple packets in data flow on a communications network.

One of the major subsystems is the Analyzer. This component is responsible for creating and maintaining classified traffic flows, processing statistics for packets and flows, managing the traffic flow database and cache, and performing state-based analysis of traffic flows.

This document describes the processes required for recognizing and maintaining state information for traffic flows. There are several different processes, which are detailed in the following sections.

11.2 Analyzer Data Input Requirements

In order for the Analyzer to successfully classify traffic by application, there are several data elements required from each packet to be analyzed. Prior to sending a packet of information to the Analyzer, all additional information must be formatted and sent along with the appropriate packet content.

The Analyzer must specifically receive each packets in a conversation in the order which they are exchange between the client and the server. The order is crucial for proper state based classification.

11.3 State-base Traffic Classification

More applications running over data networks utilize complex methods of classifying traffic through the creation of multiple states. The creation of the state based traffic classification causes the need for managing and maintaining learned states from traffic derived in the network.

There are several different methods in place for the creation of states in client/server network traffic. Even though there are several different methods for the creation of state. It is possible to isolate these different approaches into two basic categories.

The first category is commonly referred to as "server announcement". In the server announcement mode there are messages which are put out onto the network, in either a broadcast or multicast approach which, all stations in the network receive and decode to derive the appropriate connection point for communicating for that particular application, with the particular server. There are several examples for this type of server announcement implementation with state based protocols. Using the server announcement method, a particular application communicates using a service channel, in the form of a TCP or UDP socket or Port as in the IP protocol suite, or using a SAP as in the Novell IPX protocol suite.

The second category is referred to as "in-stream analysis". This method is used either as a primary or secondary recognition process. As a primary process, in-stream analysis assists in extracting detailed information which will be used to further recognize both the specific application and application component. A good example of in-stream analysis is any Web-based applications. The commonly used Pointcast Web information application can be recognized using this process. During the initial connection between a Pointcast server and client, specific key tokens exist in the data exchange that will result in a signature for Pointcast.

The in stream analysis process may also be combined with the server announcement process. In many cases in stream analysis will augment other recognition processes. An example of combining in stream analysis with server announcement can be found in business applications such as SAP and BAAN.

11.3.1 Session Tracking

One of the primary processes for tracking applications in the stream of the client/server packet exchange, is through session tracking. The process of tracking sessions requires an initial connection to a predefined

socket or Port. This method of communication is used in a variety of transport layer protocols. It is most commonly seen in the TCP and UDP transports of the IP protocol.

During the process of session tracking, a client will make the request of a server using a specific Port or socket number. This initial request will cause the server to create a TCP or UDP Port to exchange the remainder of the data between the client and the server. The server then replies to the request of the client using this newly created Port. The original Port used by the client to connect to server will never be used again during this data exchange.

One of the best examples of session tracking is TFTP. During the client/server exchange process of TFTP, a specific Port is always used to initiate the conversation. When the client begins the process of communicating, a request is made to UDP Port 67. Once the server receives this request, a new Port is created on the server. The server then replies to the client using the new Port. In this example, it is clear that in order to recognize TFTP the process must analyze the initial request from the client. Also, the reply from the server with the key Port information must be analyzed and used to create a key for monitoring the remainder of this data exchange.

Another important component in session tracking is the understanding of the current state for particular connections in the network. Many of the application protocols, which can be monitored, are transported via protocols that have built-in state information. An example of such a transport protocol is TCP. This transport provides a reliable means of sending information between a client and a server. When the data exchange is initiated a TCP request for synchronization message is sent. This message contains a specific sequence number that is used to track and acknowledgement from the server. Once the server has knowledge to the synchronization request, data is exchanged between the client and the server. When communications are no longer required, the client would send a finish or complete message to the server. The server will acknowledge this finish request, with a reply containing the sequence numbers from the request. This sequence of events is known as a connection oriented data exchange. Many of the events used to track the state in a conversation are directly related to these types of connection and maintenance messages.

All of the processes discussed above are required to track sessions. The capability to track sessions is a requirement for understanding the current state to analyze.

11.3.2 Server Announcement

The process of server announcement consists of a server with multiple applications, which are all required to be simultaneously accessed from multiple clients. Many applications are beginning to use this process as a means of multiplexing a single Port or socket into many applications and services. The individual methods of server announcement protocols tend to vary. However, the basic underlying process remains similar between all of these different announcement exchanges.

11.3.2.1 Sun RPC Analysis

Sun-RPC and Net-RPC are to good examples of server announcement oriented communications processes. In this section we will analyze the requirements for recognizing applications which utilize the sun implementation of RPC. RPC stands for remote procedure call. This is a quite clear description of the process. A remote or client that wishes to use a server or procedure must establish a connection using the RPC protocol.

Using the Sun-RPC protocol as a model for server announcement is completed through the following process. Each server running the Sun-RPC protocol must maintain a process and database called the Port Mapper. The Port Mapper creates a direct association between a Sun-RPC program or application and a TCP or UDP socket or Port. An application or program number is a 32-bit unique identifier assigned by IANA. Each Port Mapper on a Sun-RPC server can present the mappings between a unique program number and a specific transport socket through the use of specific request or a directed announcement.

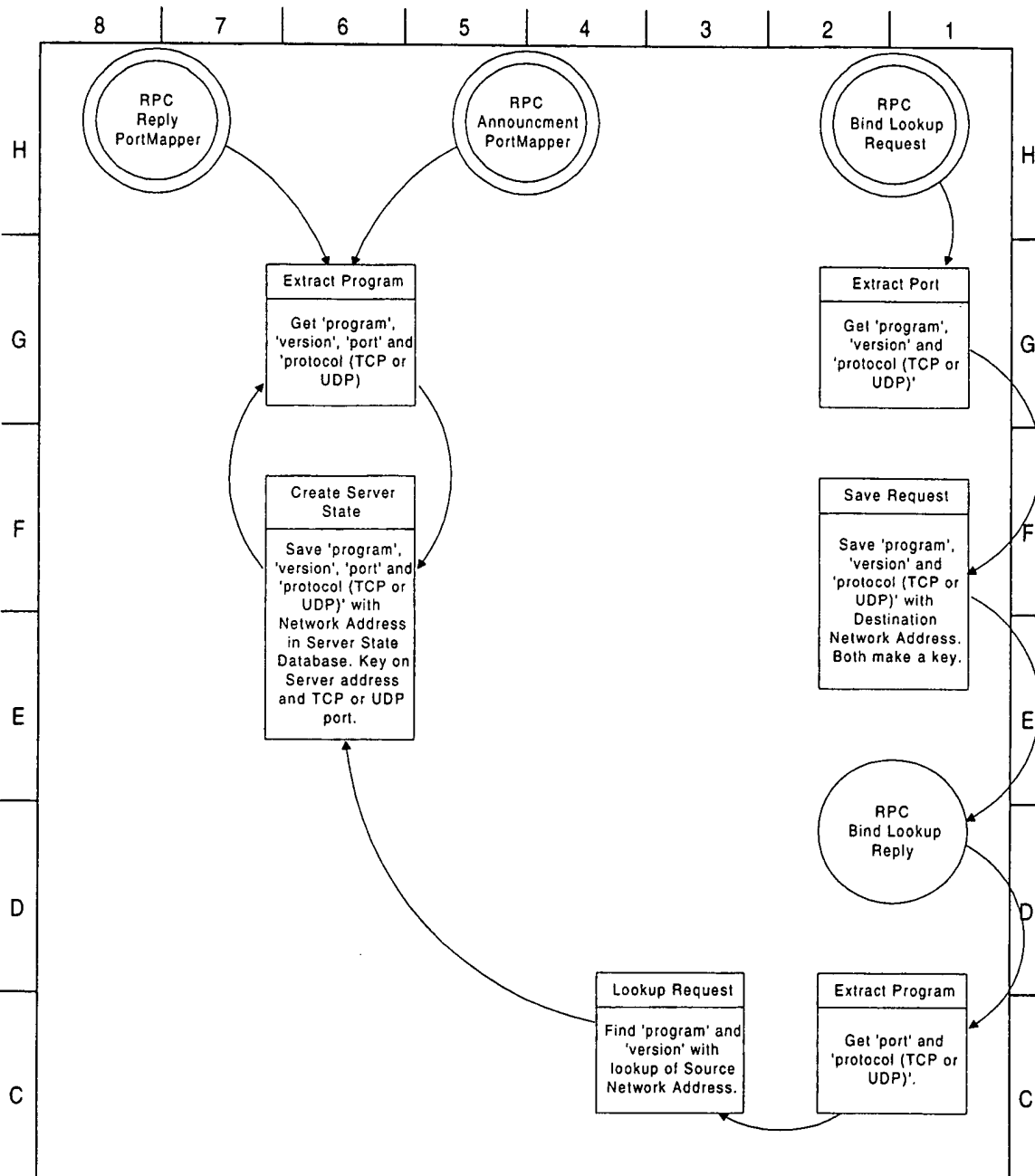
The first approach we will review is the specific request method. Using this process the client makes a specific request to the server on a predefined UDP or TCP socket. Once the Port Mapper process on the sun RPC server receives the request, the specific mapping is returned in a directed reply to the client.


- 1) A client sends a TCP packet to Port 111, with an RPC Bind Lookup Request.

- 2) The server extracts the program identifier and version identifier from the request. The server also uses the fact that this packet came in the using the TCP transport.
- 3) The server sends a TCP packet to Port 111, with an RPC Bind Lookup Reply. The reply contains the specific ports on which future transactions will be accepted for the specific RPC program identifier.

11.3.2.2 Process for Sun RPC Analysis

1. Decode Sun RPC by TCP or UDP Port 111
2. Check RPC type field for Id
3. If value is PortMapper, save paired socket (i.e. dest for dest, src for src)
4. Decode ports and mapping, save ports with socket/addr key
5. There may be more than one pairing per mapper packet
6. Saving is complete



 Technically Elite <small>mastering your enterprise</small>		TITLE FLOW - DATA FLOW	
COMPANY CONFIDENTIAL		DESCRIPTION A VISIO TEMPLATE FOR CREATING DATA FLOW DIAGRAMS USING SHAPES FROM FL_DATA.VSS.	
FILENAME		SIZE FSCM NO DWG NO REV C 7001501-STA A	
DRAWN BY COPYRIGHT © 1993 SHAPEWARE CORPORATION.		SCALE 1 : 1 DATE 04/24/98 SHEET 1 OF 1	
ALL RIGHTS RESERVED.			

PortMapper Protocol Specification (in RPC Language)

```
const PMAP_PORT = 111; /* portmapper port number */
```

A mapping of (program, version, protocol) to port number:

```
struct mapping { unsigned int prog; unsigned int vers; unsigned int prot; unsigned int port; };
```

Supported values for the "prot" field:

```
const IPPROTO_TCP = 6; /* protocol number for TCP/IP */ const IPPROTO_UDP = 17; /* protocol number for UDP/IP */ A list of mappings: struct *pmaplist { mapping map; pmaplist next; };
```

Arguments to callit: struct call_args { unsigned int prog; unsigned int vers; unsigned int proc; opaque args<>; }; Results of callit: struct call_result { unsigned int port; opaque res<>; }; Port mapper procedures: program PMAP_PROG { version PMAP_VERS { void PMAPPROC_NULL(void) = 0; bool PMAPPROC_SET(mapping) = 1; bool PMAPPROC_UNSET(mapping) = 2; unsigned int PMAPPROC_GETPORT(mapping) = 3; pmaplist PMAPPROC_DUMP(void) = 4; call_result PMAPPROC_CALLIT(call_args) = 5; } = 2; } = 100000; A.2 Port Mapper Operation The portmapper program currently supports two protocols (UDP and TCP). The portmapper is contacted by talking to it on assigned port number 111 (SUNRPC) on either of these protocols. The following is a description of each of the portmapper

Sun RPC Decode Process

- 1) Parse frame to TCP or UDP
- 2) Lookup paired sockets if no standard match
- 3) If RPC found, same new Key

The port mapper program maps RPC program and version numbers to transport-specific port numbers. This program makes dynamic binding of remote programs possible. This is desirable because the range of reserved port numbers is very small and the number of potential remote programs is very large. By running only the port mapper on a reserved port, the port numbers of other remote programs can be ascertained by querying the port mapper. The port mapper also aids in broadcast RPC. A given RPC program will usually have different port number bindings on different machines, so there is no way to directly broadcast to all of these programs. The port mapper, however, does have a fixed port number. So, to broadcast to a given program, the client actually sends its message to the port mapper located at the broadcast address. Each port mapper that picks up the broadcast then calls the local service specified by the client. When the port mapper gets the reply from the local service, it sends the reply on back to the client.

PortMapper Protocol Specification (in RPC Language)

```
const PMAP_PORT = 111; /* portmapper port number */
```

A mapping of (program, version, protocol) to port number:

```
struct mapping { unsigned int prog; unsigned int vers; unsigned int prot; unsigned int port; };
```

Supported values for the "prot" field:

```
const IPPROTO_TCP = 6; /* protocol number for TCP/IP */ const IPPROTO_UDP = 17; /* protocol number for UDP/IP */ A list of mappings: struct *pmaplist { mapping map; pmaplist next; };
```

Arguments to callit: struct call_args { unsigned int prog; unsigned int vers; unsigned int proc; opaque args<>; }; Results of callit: struct call_result { unsigned int port; opaque res<>; }; Port mapper procedures: program PMAP_PROG { version PMAP_VERS { void PMAPPROC_NULL(void) = 0; bool PMAPPROC_SET(mapping) = 1; bool PMAPPROC_UNSET(mapping) = 2; unsigned int PMAPPROC_GETPORT(mapping) = 3; pmaplist PMAPPROC_DUMP(void) = 4; call_result PMAPPROC_CALLIT(call_args) = 5; } = 2; } = 100000;

11.3.3 Port Mapper Operation

The portmapper program currently supports two protocols (UDP and TCP). The portmapper is contacted by talking to it on assigned port number 111 (SUNRPC) on either of these protocols. The following is a description of each of the portmapper procedures: PMAPPROC_NULL: This procedure does no work. By convention, procedure zero of any protocol takes no parameters and returns no results.

PMAPPROC_SET: When a program first becomes available on a machine, it registers itself with the port mapper program on the same machine. The program passes its program number "prog", version number "vers", transport protocol number "prot", and the port "port" on which it awaits service request. The procedure returns a boolean reply whose value is "TRUE" if the procedure successfully established the mapping and "FALSE" otherwise. The procedure refuses to establish a mapping if one already exists for the tuple "(prog, vers, prot)".

PMAPPROC_UNSET: When a program becomes unavailable, it should unregister itself with the port mapper program on the same machine. The parameters and results have meanings identical to those of "PMAPPROC_SET". The protocol and port number fields of the argument are ignored.

PMAPPROC_GETPORT: Given a program number "prog", version number "vers", and transport protocol number "prot", this procedure returns the port number on which the program is awaiting call requests. A port value of zeros means the program has not been registered. The "port" field of the argument is ignored.

PMAPPROC_DUMP: This procedure enumerates all entries in the port mapper's database. The procedure takes no parameters and returns a list of program, version, protocol, and port values.

PMAPPROC_CALLIT: This procedure allows a client to call another remote procedure on the same machine without knowing the remote procedure's port number. It is intended for supporting broadcasts to arbitrary remote programs via the well-known port mapper's port. The parameters "prog", "vers", "proc", and the bytes of "args" are the program number, version number, procedure number, and parameters of the remote procedure. Note: (1) This procedure only sends a reply if the procedure was successfully executed and is silent (no reply) otherwise. (2) The port mapper communicates with the remote program using UDP only. The procedure returns the remote program's port number, and the reply is the reply of the remote procedure.

11.3.4 Service Announcement

Service announcement method of the application recognition is very similar to server announcement. One specific difference in service announcement is that the announcements are made regularly and contain fixed information. Also, service announcement based applications only provide the key information for locating applications in each announcement. There is no capability to request a specific service. Each client must learn the key information required to access an application.

Novell's IPX SAP is a good example of service announcement oriented communications process. A Novell server will have many different services, which it may provide to clients on network. IPX uses service access points or SAP as a way to identify specific applications and services.

11.3.5 In-stream Recognition and Extraction

The process of identifying more of the business applications on networks today requires analysis of information in stream of the network data. Simply, this means that in order to contain the visibility to application traffic flow, a process must routinely analyze the network stream itself.

SMB is a protocol used to in networks today which has textual information during the data exchange that can be used to further determine the type of end-user application involved in communications. An SMB packet is usually transported above the NetBIOS session protocol. Inside the SMB header is a function code. This function code is one octet in length and assists in the classification of the type of SMB data in the payload.

11.3.5.1 Web-based Applications

The best example of applications requiring in-stream recognition mainly Web-based. These applications generally utilize two well-known ports for all conversations. Because of this, they can be considered multiplex ports. There is one big difference, the client and server have no well-known exchange mechanism outside of the normal data stream. Therefore, these applications require combining session tracking and in stream recognition to derive end-user application.

As discussed earlier, point cast is one of the most widely used Web based application. The steps required to detail point cast can be repeated for other Web based applications. This is also a good example for understanding the process used in combining session tracking with other recognition techniques.

The process begins when a client Web browser initiates a request to a point cast Web server. This request



- **Exhibit A4:** Protocol Tracking Summary (Document MFAProtocolLayout.pdf)

Description	Type	TCP/IP Access			Novell Access			Unix			Misc	
		Well			Well			Well			New	
		Content	HTTP	Pattern	Well	Known	UDP	Other	Well	Known	Well	Other
2.0 Alternate HTTP	Well-Known	HTTP	Well		591, 8008, 8080							
14.0 Sybase Adaptive SQL Anywhere (ASA) Server	Well-Known	HTTP	Known		1498, 2638	0x00c5					Sybase/Mcgr	
14.2 Tabular Domain Stream (TDS)	Well-Known	HTTP	Known			0x00c5					Sybase/ASA	
15.0 Command Sequence (CmdSeq)	Well-Known	HTTP	Known									
15.0 Sybase Adaptive SQL Enterprise (ASE) Server	Well-Known	HTTP	Known									
16.0 Microsoft SQL Server	Well-Known	HTTP	Known									
17.0 DB2	Well-Known	HTTP	Known									
19.0 Cisco Dynamic ISL (DSL)	Well-Known	HTTP	Known									
19.0 Cisco gateway Discovery Protocol (CDP)	Well-Known	HTTP	Known									
20.0 Novell IPXWAN (RFC 1634)	Well-Known	HTTP	Known									
27.1 MS-Exchange - POP3 Mail	Well-Known	HTTP	Known									
27.1 MS-Exchange - SMTP Mail	Well-Known	HTTP	Known									
27.1 MS-Exchange - IMAP4 Mail	Well-Known	HTTP	Known									
27.1 MS-Exchange - LDAP	Well-Known	HTTP	Known									
27.1 MS-Exchange - ISO over TCP/IP	Well-Known	HTTP	Known									
27.1 MS-Exchange - X.400	Well-Known	HTTP	Known									
27.1 MS-Exchange - DCE Endpoint Mapping	Well-Known	HTTP	Known									
32.0 Vines Token-Ring (vtr)	Well-Known	HTTP	Known									
33.0 SMTP over SSL	Well-Known	HTTP	Known									
33.0 NNTP over SSL	Well-Known	HTTP	Known									
33.0 Shell over SSL	Well-Known	HTTP	Known									
33.0 LDAP over SSL	Well-Known	HTTP	Known									
33.0 FTP-data over SSL	Well-Known	HTTP	Known									
33.0 FTP-control over SSL	Well-Known	HTTP	Known									
33.0 Telnet over SSL	Well-Known	HTTP	Known									
33.0 IMAP4 over SSL	Well-Known	HTTP	Known									
33.0 IRC over SSL	Well-Known	HTTP	Known									
33.0 POP3 over SSL	Well-Known	HTTP	Known									
34.0 VPN - PPTP	Well-Known	HTTP	Known									
35.0 Cu-SeeMe	Well-Known	HTTP	Known									
37.0 PcAnywhere	Well-Known	HTTP	Known									
38.0 Timbuktu	Well-Known	HTTP	Known									
39.1 StreamWorks	Well-Known	HTTP	Known									
40.0 VDOlive	Well-Known	HTTP	Known									
41.0 FreeTel	Well-Known	HTTP	Known									
42.0 Microsoft System Mgmt Server	Well-Known	HTTP	Known									
43.0 Microsoft Message Queue Service	Well-Known	HTTP	Known									
44.0 Distributed.net	Well-Known	HTTP	Known									
45.0 OpenWindows	Well-Known	HTTP	Known									
46.0 XWindows (X.11)	Well-Known	HTTP	Known									
47.0 America Online (AOL)	Well-Known	HTTP	Known									
48.0 talk	Well-Known	HTTP	Known									
48.0 ntalk	Well-Known	HTTP	Known									
48.0 Internet Relay Chat (IRC)	Well-Known	HTTP	Known									
50.0 iChat	Well-Known	HTTP	Known									
51.0 iVist	Well-Known	HTTP	Known									
52.0 The Palace	Well-Known	HTTP	Known									
53.0 Network Time Protocol (NTP)	Well-Known	HTTP	Known									
54.0 TACACS	Well-Known	HTTP	Known									
55.0 Netbios SSN over TCP	Well-Known	HTTP	Known									
56.0 Netbios NIM over UDP	Well-Known	HTTP	Known									
57.0 SMB over Netbios Session	Well-Known	HTTP	Known									
60.0 Quake/Quake-II	Well-Known	HTTP	Known									
61.0 QuakeWorld	Well-Known	HTTP	Known									

Description	Type	TCP/IP Access				Novell Access		Well			Unix	DceRPC	Misc		
		HTTP Content	HTTP Pattern	Well Known TCP	Well Known UDP	Other	Well Known IPX/SPX	SAP Mapped	Well Known VIP/NSPP	Well Known VIPC	Other	Port Mapped	Endpoint Mapped	New Known Parents	Other
3.0 RealAudio	State Based	.	.	7070	6970-7170	CORE #1 core #1									CORE #10 core #10
4.0 Podcast	State Based	.	.		370	CORE #4 CORE #7									core #10 core #10
5.0 BackWeb	State Based	.	.	1755		core #1 core #1									core #10 core #10
6.0 Microsoft Media (formerly NetShow)	State Based	.	.			core #1 core #1									core #10 core #10
7.0 QuickTime	State Based	.	.			core #1 core #1									core #10 core #10
8.0 VivoActive	State Based	.	.			core #1 core #1									core #10 core #10
9.0 Shockwave	State Based	.	.			core #1 core #1									core #10 core #10
10.0 PowerBuilder (Sybase/Powersoft)	State Based	.	.			core #1 core #1									core #10 core #10
11.0 Web.SQL (Sybase)	State Based	.	.			core #1 core #1									core #10 core #10
12.0 iConnect/JDBC (Sybase)	State Based	.	.	1521, 1526, 1527											core #10 core #10
18.2 Oracle - Transparent Network Substrate	State Based	.	.												core #10 core #10
18.3 SQL*Net	State Based	.	.												core #10 core #10
18.3 MS-ODBC	State Based	.	.												core #10 core #10
18.3 PeopleSoft	State Based	.	.												core #10 core #10
18.3 SAP	State Based	.	.												core #10 core #10
21.0 IP-fragmentation	State Based	.	.			CORE #3 CORE #2									core #10 core #10
22.0 SunRPC PortMapper	State Based	.	.			CORE #2 core #2									core #10 core #10
23.0 Mount	State Based	.	.		2049	core #2 core #2									core #10 core #10
24.0 NFS	State Based	.	.			core #2 core #2									core #10 core #10
25.0 Yellow Pages	State Based	.	.			core #2 core #2									core #10 core #10
25.0 db Session Manager	State Based	.	.			core #2 core #2									core #10 core #10
25.0 pcNFS	State Based	.	.			core #2 core #2									core #10 core #10
25.0 3270_mapper	State Based	.	.			core #2 core #2									core #10 core #10
25.0 file_mapper	State Based	.	.			core #2 core #2									core #10 core #10
25.0 NIS+ (National Information Service)	State Based	.	.			core #2 core #2									core #10 core #10
25.0 rsat	State Based	.	.												core #10 core #10
26.0 Novell SAP	State Based	.	.												core #10 core #10
27.2 MS-Exchange - Information Store	State Based	.	.												core #10 core #10
27.2 MS-Exchange - Directory	State Based	.	.												core #10 core #10
27.2 MS-Exchange - MTA	State Based	.	.												core #10 core #10
28.6 DceRPC Endpoint Mapper (conn-less)	State Based	.	.												core #10 core #10
28.7 DceRPC Endpoint Mapper (conn-oriented)	State Based	.	.	135	135										core #10 core #10
29.0 Vines IPC-RDP	State Based	.	.												core #10 core #10
30.0 Vines SMB over SPP	State Based	.	.												core #10 core #10
31.0 Vines Print	State Based	.	.												core #10 core #10
31.0 Vines Async	State Based	.	.												core #10 core #10
36.0 Citrix	State Based	.	.	1494	1604	777 tcp - tbd tbd*....	0x85ba-859b	0x052d, 0x083d							core #11 tbd
58.0 Microsoft NetMeeting	State Based	.	.												core #11 tbd
59.0 X.400	State Based	.	.												core #11 tbd

CORE #1 - HTTP Engine
 CORE #2 - SunRPC PortMapper Engine
 CORE #3 - IP Fragmentation Engine
 CORE #4 - BackWeb UDP Engine
 CORE #5 - Vines IPC Engine
 CORE #6 - Vines SPP Engine
 CORE #7 - MS Media Engine
 CORE #8 - DceRPC CO Mapper Engine
 CORE #9 - DceRPC CL Mapper Engine
 CORE #10 - Oracle TNS Engine
 CORE #11 - Novell SAP Engine

TCP port 80, connection-oriented, STATE-BASED core for HTTP
 UDP port 111, connection-less, STATE-BASED core for PortMapper
 IP Fragmentation STATE-BASED core mapping fragments #2 - n to their flows from fragments #1
 UDP port 370, connection-less, STATE-BASED core to map responses to requests on port #370
 VIPC-RDP connection-oriented, STATE-BASED core to track VIPC sessions (for all sessions)
 VSP connection-oriented, STATE-BASED core to track VSP sessions (for non-well-known sockets)
 TCP port 1755, connection-oriented, STATE-BASED core to handle dynamic UDP Port Assignment
 connection-oriented, STATE-BASED core for DCE RPC Endpoint Mapping
 connection-less STATE-BASED core for DCE RPC Endpoint Mapping
 TCP port 1521/1526/1527, connection oriented, STATE-BASED core for Oracle TNS session tracking
 connection-less, STATE-BASED core for Service Advertisement Program (SAP) mapping

Description	Type	HTTP		Well Known TCP	Well Known UDP	Other	Well Known IPX/SPX	SAP Mapped	Well Known VIP/SP	Well Known VIPC	Other	Port Mapped	Endpoint Mapped	New Parents	Other
		Content Type	HTTP Pattern												
3.0 plain	State Based	text				core #1									
3.0 html	State Based	text				core #1									
3.0 tab-separated-values	State Based	text				core #1									
3.0 sgml	State Based	text				core #1									
3.0 rich-text	State Based	text				core #1									
3.0 enriched	State Based	text				core #1									
3.0 real-audio	State Based	text				core #1									
3.0 gif	State Based	image				core #1									
3.0 jpeg	State Based	image													
3.0 pict	State Based	image													
3.0 x-bitmap	State Based	image													
3.0 x-pixmap	State Based	image													
3.0 cgm	State Based	image													
3.0 group-3-lax	State Based	image													
3.0 png	State Based	image													
3.0 tcl	State Based	image													
3.0 tiff	State Based	image													
3.0 real-audio	State Based	image													
3.0 quicktime	State Based	image													
3.0 basic-audio	State Based	audio													
3.0 AIFF	State Based	audio													
3.0 mpeg2-audio	State Based	audio													
3.0 WAV	State Based	audio													
3.0 real-audio	State Based	audio													
3.0 MIDI	State Based	audio													
3.0 x-windows-dump-image	State Based	audio													
3.0 mpeg-audio-3	State Based	audio													
3.0 vml	State Based	x-world													
3.0 quicktime	State Based	applic													
3.0 mpeg2-video	State Based	applic													
3.0 sgi-video	State Based	applic													
3.0 real-audio	State Based	applic													
3.0 ms-media	State Based	applic													
3.0 avi	State Based	applic													
3.0 vivo-active	State Based	applic													
3.0 mac-binhex40	State Based	applic													
3.0 mac-stuffit	State Based	applic													
3.0 mac-binary	State Based	applic													
3.0 compress	State Based	applic													
3.0 zip	State Based	applic													
3.0 gzip	State Based	applic													
3.0 tar	State Based	applic													
3.0 postix-lar	State Based	applic													
3.0 gnu-lar	State Based	applic													
3.0 cpio	State Based	applic													
3.0 bcpio	State Based	applic													
3.0 c-shell	State Based	applic													
3.0 bourne-shell	State Based	applic													
3.0 tcl	State Based	applic													
3.0 octet-stream	State Based	applic													
3.0 javascript	State Based	applic													
3.0 mpeg-audio-3	State Based	applic													
3.0 rich-text	State Based	applic													
3.0 tex	State Based	applic													
3.0 latex	State Based	applic													
3.0 tex-dvi	State Based	applic													
3.0 gnu-texinfo	State Based	applic													
3.0 oda	State Based	applic													

Protocol Tracking Summary

3.0 edifact	State Based	applic					
3.0 edi-x12	State Based	applic					
3.0 edi-consent	State Based	applic					
3.0 news	State Based	applic					
3.0 microsoft-word	State Based	applic					
3.0 microsoft-excel	State Based	applic					
3.0 microsoft-powerpoint	State Based	applic					
3.0 microsoft-project	State Based	applic					
3.0 lotus-organizer	State Based	applic					
3.0 lotus-free lance	State Based	applic					
3.0 lotus-123	State Based	applic					
3.0 lotus-approach	State Based	applic					
3.0 lotus-wordpro	State Based	applic					
3.0 troll	State Based	applic					
3.0 wordperfect	State Based	applic					
3.0 quattro-pro	State Based	applic					
3.0 framemaker	State Based	applic					
3.0 postscript	State Based	applic					
3.0 visio	State Based	applic					
3.0 sgml	State Based	applic					
3.0 powerbuilder	State Based	applic					
3.0 real-audio	State Based	applic					
3.0 shockwave	State Based	applic					
3.0 acrobat	State Based	applic					
3.0 lotus-notes	State Based	applic					

- **Exhibit B0** is a dated computer directory of test data and documents used therefore.

BEST AVAILABLE COPY

Directory of M:\aaa-----INVENTEK CLIENTS\Hifn\Patents\APPT-001-1-1 filed
Proof of Reductn to Practice\Compiler\

M:\aaa-----INVENTEK CLIENTS\Hifn\Patents\APPT-001-1-1 filed Proof of
Reductn to Practice\Compiler\

```
=====
big.cpl                548 KB      04:17:18 AM      a
bigfgc3.cpl           1164 KB      04:06:18 PM      a
bigfgpc.cpl           1164 KB      04:06:18 PM      a
bigfpay1.cpl          1051 KB      09:57:44 AM      a
bigfpay12.cpl         1054 KB      10:17:18 AM      a
bigfpgrp.cpl          1159 KB      11:04:40 AM      a
bigfpgrp2.cpl         1163 KB      10:11:06 AM      a
bigfrag.cpl           995 KB      07:17:34 AM      a
bigfrag2.cpl          999 KB      10:21:52 AM      a
mfaptkey.txt           1 KB      03:05:54 PM      a
mfaptkey2.txt          1 KB      07:54:12 AM      a
mfaptpkt.txt           4 KB      03:07:00 PM      a
mfaptpkt2.txt          4 KB      01:52:42 AM      a
MFATEST.HEX           213 KB      02:53:04 PM      a
MFATEST.TXT            70 KB      03:00:48 PM      a
MFS-PDL-Reference.pdf  97 KB      04:10:18 AM      a
MFS-State-Classification.pdf 121 KB      04:11:28 AM      a
output.cpl            209 KB      08:45:34 AM      a
packets.txt           46 KB      09:29:04 AM      a
Protocols.cpl         204 KB      10:12:10 AM      a
short.cpl             150 KB      08:38:42 AM      a
shrtfpg2.cpl          290 KB      10:14:38 AM      a
shrtfps3.cpl          256 KB      02:25:12 PM      a
shrtfps4.cpl           86 KB      10:35:56 AM      a
shrtfps5.cpl           86 KB      10:35:56 AM      a
shrttun1.cpl          171 KB      12:21:42 PM      a
=====
```

AA

Total 0 folder(s); 26 file(s)

Total files size: 11 MB; 11315 KB; 11586502 Bytes

AA

- **Exhibit B1:** Technically Elite MeterFlow Accelerator Modules Testbench Specification (Document MFATest.pdf in directory of Exhibit A0)

DRAFT

Technically Elite MeterFlow Accelerator Modules Testbench Specification

Not For External Release!

Revision History		
Version	Date	Description
0.1	[REDACTED]	Collect earlier documents. Format document.
0.9	[REDACTED]	Technically Elite review release.

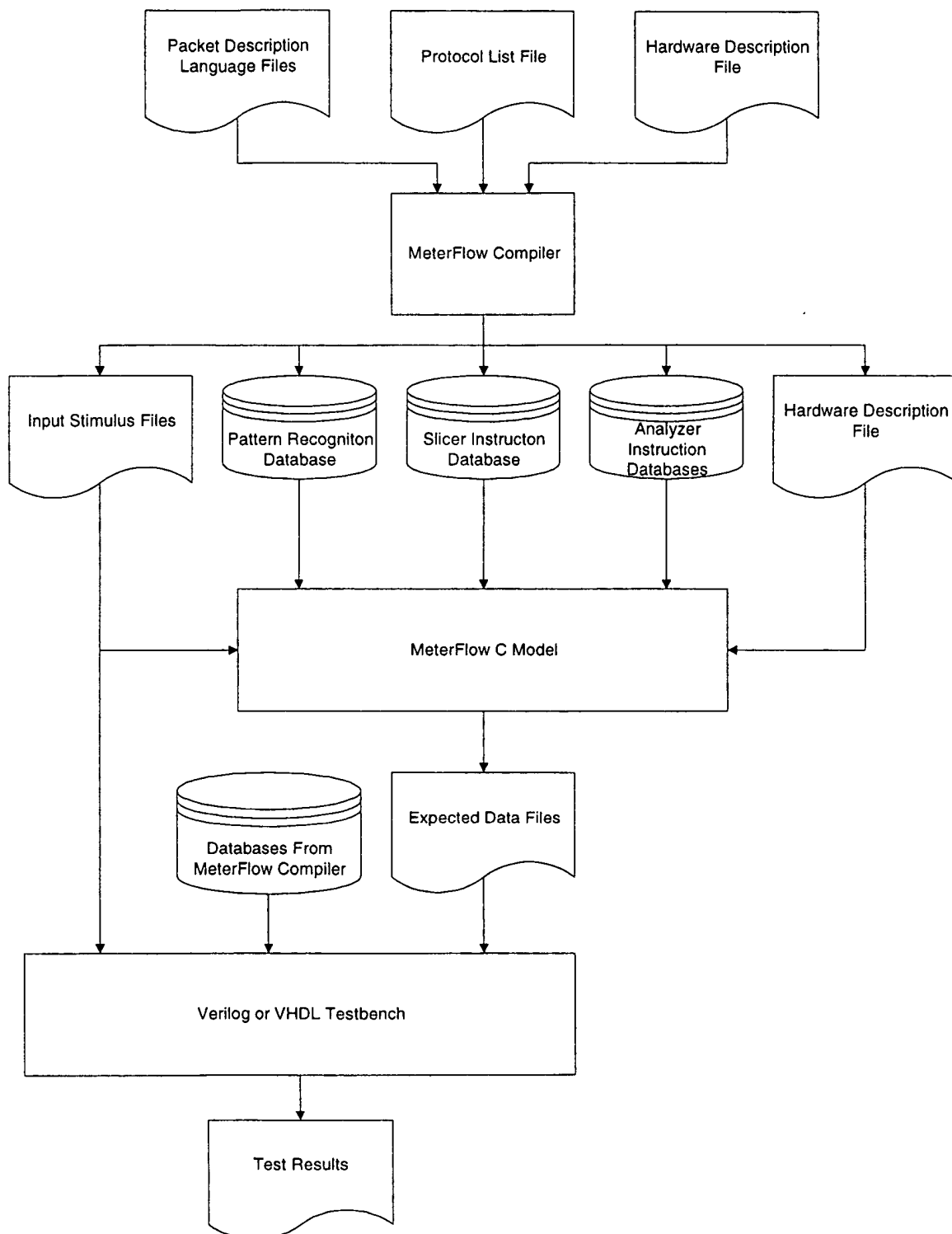
1 Introduction

This document describes the methodology to be used to build testbenches for the MeterFlow Accelerator Modules Verilog and VHDL implementations. The goal is to have fully automated testing. This means that the unit under tests (UUT) output is compared to expected data generated by the C model and the results can be reported as pass/fail. The input to the testbenches are files generated by the MeterFlow Compiler.

1.1 Technically Elite MeterFlow Accelerator Modules Testbench

- Written in both the Verilog and VHDL
- Asynchronous interfaces each have a separate clock
- Automated testing and result reporting
- The same input files read by the testbenches and the C model

2 Test Flow Chart



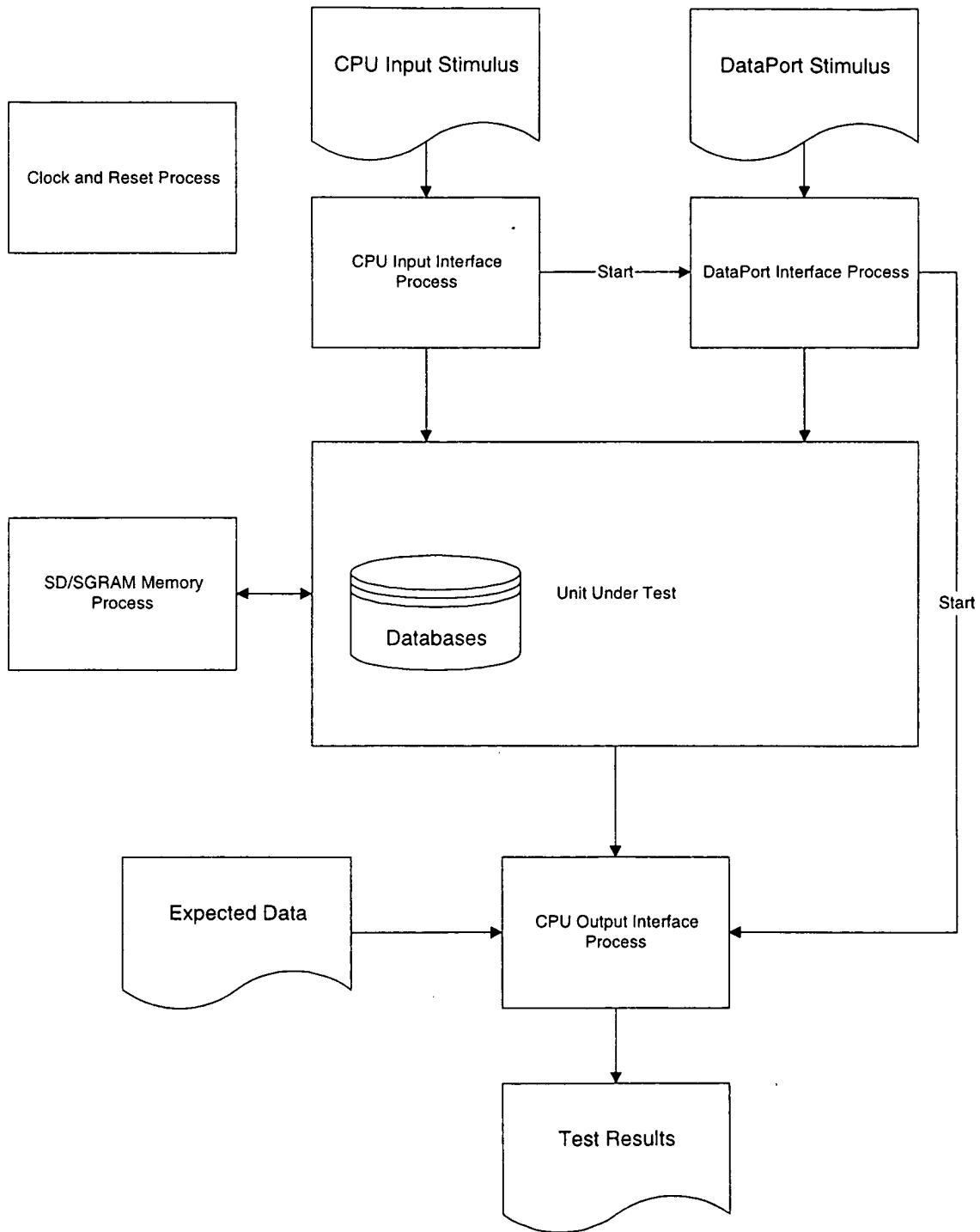
2.1 Test Flow Chart Description

The MeterFlow Compiler takes as its input three sets of files. The first is the Protocol List file. This file describes the protocols this implementation of the hardware must recognize and process. The compiler will then lookup each of the protocols in the list for their Packet Description Language file. Each of these files describes how to recognize and process the protocol. Finally the compiler may be given a Hardware Description files that specifies the hardware resources available for this implementation. The compiler can also generate this file by determining the minimum resources required to implement all the protocols in the list.

The compiler outputs the databases used by the MeterFlow accelerator. These databases can be read into the C model, the UUT and the actual hardware (if it exists). It also outputs a set of input stimulus files for both the C model and the testbenches.

The C model emulates the functions of the UUT and produces expected data files. These files contain cycle by cycle data that the testbench uses to check the results of the test.

3 Testbench Block Diagram



3.1 Testbench Block Diagram Description

The testbenches are built up of separate processes run concurrently. The Clock and Reset Process generates the system clock and system reset signals. If a process requires a different clock, such as the SD/SGRAM Memory Process, it generates that clock itself.

The SD/SGRAM Memory Process instantiates the target memory the system is to use. An accurate model of the memory is required to assure valid results.

The three processes that are shown importing files, each instantiate memories to hold the data read from the files. These memories act as patterns to be either driven into the UUT or patterns the output of the UUT are compared against. Since the UUT must be programmed before testing can begin, there is a handshake between each of the three processes. This is shown in the diagram as the Start signals.

The test begins with the CPU Input Interface Process programming the UUT. Once the UUT is programmed, the CPU Input Interface Process raises it's Start output. This tells the DataPort Interface Process to begin sending packets into the UUT. After the packets are completed the DataPort Interface Process raises it's Start output. The CPU Output Interface Process then begins reading the flow database. It checks the flows against the expected data and writes the Test Results file.

- **Exhibit B2:** The first page of file big.cpl.

The cpl files (big.cpl, bigfgc3.cpl, bigfgpc.cpl, bigfpayl.cpl, bigfpayl2.cpl, bigfpgrp.cpl, bigfpgrp2.cpl, bigfrag.cpl, bigfrag2.cpl, output.cpl, Protocols.cpl, short.cpl, shrtfpg2.cpl, shrtfps3.cpl, shrtfps4.cpl, shrtfps5.cpl, shrttunl.cpl) are files for the protocol compiler of all the actual protocols recognized by the system. These files include a description of the parser information for the parser to perform the parsing/extracting operation according to the protocol. They also contain the state processing states for the state operations of elements (d) and (e) of claim 54. The first page of one file is provided.

-- Generated on 22:04:05

0x017C -- Total number of protocols (380 dec)

--

-- Virtual Layer Decodes

--

VirtualBase -- Text Name

0x00 -- InternalProtocolCode
0x01 -- HeaderLengthFixed (0x00 - no [computed], 0x01 - yes [fixed])
0x00 -- HeaderLengthElementSize (0x00 - byte, 0x01 nibble)
0x00 -- HeaderLengthWord (0x00 - byte count, 0x01 word count (32 bits))
0x01 -- HeaderLengthField (byte offset or nibble offset)
0x00 -- DLCLayerFlag (NO)
0x00 -- DLCLayerDestOffset (NULL)
0x00 -- DLCLayerDestMask (NULL)
0x00 -- DLCLayerSrcOffset (NULL)
0x00 -- DLCLayerSrcMask (NULL)
0x00 -- NetLayerFlag (NO)
0x00 -- NetLayerAddressSize (NULL)
0x00 -- NetLayerDestOffset (NULL)
0x00 -- NetLayerDestMask (NULL)
0x00 -- NetLayerSrcOffset (NULL)
0x00 -- NetLayerSrcMask (NULL)
0x00 -- NetLayerFragments (NULL)
0x00 -- TunnelLayerFlag (NO)
0x00 -- TunnelLayerAddressSize (NULL)
0x00 -- TunnelLayerDestOffset (NULL)
0x00 -- TunnelLayerDestMask (NULL)
0x00 -- TunnelLayerSrcOffset (NULL)
0x00 -- TunnelLayerSrcMask (NULL)
0x00 -- TunnelLayerFragments (NULL)
0x00 -- ConnectionLayerFlag
0x00 -- ChildRecognitionTypeLengthFlag
0x01 -- ChildRecognitionIgnoreSource (0x00 - no, 0x01 - yes [ignore])
0x01 -- ChildRecognitionSize
0x00 -- ChildRecognitionDestOffset
0x00 -- ChildRecognitionSrcOffset
0x01 -- NumChildren (1 children)
0x01 -- RecognitionCode
0x01 -- Ethernet Base

--

-- DLC Layer Decodes

--

-- DLC (base) Ethernet V2 Decodes

EtherType -- Text Name

0x01 -- InternalProtocolCode
0x01 -- HeaderLengthFixed (0x00 - no [computed], 0x01 - yes [fixed])
0x00 -- HeaderLengthElementSize (0x00 - byte, 0x01 nibble)
0x00 -- HeaderLengthWord (0x00 - byte count, 0x01 word count (32 bits))
0x0E -- HeaderLengthField (byte offset or nibble offset)
0x01 -- DLCLayerFlag (YES)
0x00 -- DLCLayerDestOffset (0 - 5)
0xFF -- DLCLayerDestMask (All bits)
0x06 -- DLCLayerSrcOffset (6 - 11)

- **Exhibit B3:** The file MFATEST.HEX that contains the actual packets captured by the packet acquisition device described in element (a) of claims 11 and 54, and corresponding to the contents of element (b), the input buffer memory of claim 29. The packet acquisition device for the experiment was a SUN workstation connected to a connection point of a network.

42
08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00
00 30 B0 6C 40 00 80 06 94 14 59 06 06 03 59 07
FE 36 09 53 00 6E 1A 5D 8A 6E 50 DA 49 60 50 18
1D 4B BF 97 00 00 52 45 54 52 20 32 0D 0A FD 6E
9D F5

--*-*-*
00 00 00 00 08 00 20 13 10 D2 00 A0 24 75 C7 78
00 08 59 07 FE 36 00 00 00 00 00 00 00 00 00 00
00 00 59 06 06 03 00 00 00 00 00 00 00 00 00
00 00 00 2B 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 47 00 6E 00 00 09 53 00 00
--*-*-*

4B
00 A0 24 75 C7 78 08 00 20 13 10 D2 08 00 45 00
00 39 16 03 00 00 3C 06 B2 75 59 07 FE 36 59 06
06 03 00 6E 09 53 50 DA 49 60 1A 5D 8A 76 50 18
10 00 5D 6C 00 00 2B 4F 4B 20 31 33 35 31 20 6F
63 74 65 74 73 0D 0A CA E0 6A B1

--*-*-*
00 00 00 00 00 A0 24 75 C7 78 08 00 20 13 10 D2
00 08 59 06 06 03 00 00 00 00 00 00 00 00 00
00 00 59 07 FE 36 00 00 00 00 00 00 00 00 00 00
00 00 00 2B 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 47 09 53 00 00 00 6E 00 00
--*-*-*

40
08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00
00 28 B1 6C 40 00 80 06 93 1C 59 06 06 03 59 07
FE 36 09 53 00 6E 1A 5D 8A 76 50 DA 49 71 50 10
1D 3A 93 73 00 00 00 00 00 00 00 00 02 03 21 C2

--*-*-*
00 00 00 00 08 00 20 13 10 D2 00 A0 24 75 C7 78
00 08 59 07 FE 36 00 00 00 00 00 00 00 00 00 00
00 00 59 06 06 03 00 00 00 00 00 00 00 00 00
00 00 00 2B 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 47 00 6E 00 00 09 53 00 00
--*-*-*

4 TO: 00A02475C778
FROM: 0800201310D2

22:53:51<0.002>

Pkt: 4, Len: 1120/1390

0000	00 A0 24 75 C7 78 08 00	20 13 10 D2 08 00 45 00	..\$u.x... ..E.
0010	05 5C 16 05 00 00 3C 06	AD 50 59 07 FE 36 59 06	.\....<..PY..6Y.
0020	06 03 00 6E 09 53 50 DA	49 71 1A 5D 8A 76 50 18	...n.SP.Iq.].vP.
0030	10 00 53 11 00 00 52 65	74 75 72 6E 2D 50 61 74	..S...Return-Pat
0040	68 3A 20 3C 6A 6D 65 74	7A 67 65 72 40 74 65 63	h: <jmetzger@tec
0050	65 6C 69 74 65 2E 63 6F	6D 3E 0D 0A 52 65 63 65	elite.com>..Rece
0060	69 76 65 64 3A 20 66 72	6F 6D 20 6E 61 74 61 64	ived: from natad
0070	6D 2E 74 65 63 65 6C 69	74 65 2E 63 6F 6D 20 62	m.tecelite.com b
0080	79 20 73 75 70 65 72 2E	74 65 63 65 6C 69 74 65	y super.tecelite
0090	2E 63 6F 6D 20 28 34 2E	31 2F 53 4D 49 2D 34 2E	.com (4.1/SMI-4.
00A0	31 29 0D 0A 09 69 64 20	41 41 32 38 34 30 38 3B	1)...id AA28408;
00B0	20 54 68 75 2C 20 31 30	20 53 65 70 20 39 38 20	Thu, 10 Sep 98
00C0	31 37 3A 33 37 3A 33 37	20 50 44 54 0D 0A 52 65	17:37:37 PDT..Re
00D0	63 65 69 76 65 64 3A 20	66 72 6F 6D 20 73 6D 74	ceived: from smt
00E0	70 6C 69 6E 6B 2E 74 65	63 65 6C 69 74 65 2E 63	plink.tecelite.c
00F0	6F 6D 20 28 73 6D 74 70	6C 69 6E 6B 20 5B 38 39	om (smtplink [89
0100	2E 37 2E 37 2E 31 30 30	5D 29 0D 0A 09 62 79 20	.7.7.100)}...by

0110	6E 61 74 61 64 6D 2E 74	65 63 65 6C 69 74 65 2E	natadm.tecelite.
0120	63 6F 6D 20 28 38 2E 38	2E 37 2F 38 2E 38 2E 37	com (8.8.7/8.8.7
0130	29 20 77 69 74 68 20 53	4D 54 50 20 69 64 20 52) with SMTP id R
0140	41 41 31 37 32 34 35 3B	0D 0A 09 54 68 75 2C 20	AA17245;...Thu,
0150	31 30 20 53 65 70 20 31	39 39 38 20 31 37 3A 33	10 Sep 1998 17:3
0160	39 3A 30 34 20 2D 30 37	30 30 0D 0A 52 65 63 65	9:04 -0700..Rece
0170	69 76 65 64 3A 20 66 72	6F 6D 20 63 63 3A 4D 61	ived: from cc:Ma
0180	69 6C 20 62 79 20 73 6D	74 70 6C 69 6E 6B 2E 74	il by smtpplink.t
0190	65 63 65 6C 69 74 65 2E	63 6F 6D 0D 0A 09 69 64	ecelite.com...id
01A0	20 41 41 39 30 35 34 37	34 38 32 39 20 54 68 75	AA905474829 Thu
01B0	2C 20 31 30 20 53 65 70	20 39 38 20 31 37 3A 34	, 10 Sep 98 17:4
01C0	37 3A 30 39 20 50 44 54	0D 0A 44 61 74 65 3A 20	7:09 PDT..Date:
01D0	54 68 75 2C 20 31 30 20	53 65 70 20 39 38 20 31	Thu, 10 Sep 98 1
01E0	37 3A 34 37 3A 30 39 20	50 44 54 0D 0A 46 72 6F	7:47:09 PDT..Fro
01F0	6D 3A 20 4A 6F 68 6E 20	4D 65 74 7A 67 65 72 20	m: John Metzger
0200	3C 6A 6D 65 74 7A 67 65	72 40 74 65 63 65 6C 69	<jmetzger@teceli
0210	74 65 2E 63 6F 6D 3E 0D	0A 45 6E 63 6F 64 69 6E	te.com>..Encodin
0220	67 3A 20 33 32 34 20 54	65 78 74 0D 0A 4D 65 73	g: 324 Text..Mes
0230	73 61 67 65 2D 49 64 3A	20 3C 39 38 30 38 31 30	sage-Id: <980810
0240	39 30 35 34 2E 41 41 39	30 35 34 37 34 38 32 39	9054.AA905474829
0250	40 73 6D 74 70 6C 69 6E	6B 2E 74 65 63 65 6C 69	@smtpplink.teceli
0260	74 65 2E 63 6F 6D 3E 0D	0A 54 6F 3A 20 62 6C 65	te.com>..To: ble
0270	61 76 79 40 74 65 63 65	6C 69 74 65 2E 63 6F 6D	avy@tecelite.com
0280	2C 20 61 63 68 61 64 64	61 40 74 65 63 65 6C 69	, achadda@teceli
0290	74 65 2E 63 6F 6D 2C 20	64 61 76 65 63 40 74 65	te.com, davec@te
02A0	63 65 6C 69 74 65 2E 63	6F 6D 2C 0D 0A 20 20 20	celite.com,..
02B0	20 20 20 20 20 44 61 76	69 64 20 4C 75 6F 20 3C	David Luo <
02C0	64 6C 75 6F 40 74 65 63	65 6C 69 74 65 2E 63 6F	dluo@tecelite.co
02D0	6D 3E 2C 20 6C 6F 77 64	65 72 40 74 65 63 65 6C	m>, lowder@tecel
02E0	69 74 65 2E 63 6F 6D 2C	0D 0A 20 20 20 20 20 20	ite.com,..
02F0	20 20 65 77 68 65 65 6C	65 72 40 74 65 63 65 6C	ewheeler@tecel
0300	69 74 65 2E 63 6F 6D 2C	20 66 6E 6F 6F 6E 40 74	ite.com, fnoon@t
0310	65 63 65 6C 69 74 65 2E	63 6F 6D 2C 20 66 72 65	ecelite.com, fre
0320	64 6D 40 74 65 63 65 6C	69 74 65 2E 63 6F 6D 2C	dm@tecelite.com,
0330	0D 0A 20 20 20 20 20 20	20 20 6A 6D 61 69 78 6E	.. jmaixn
0340	65 72 40 74 65 63 65 6C	69 74 65 2E 63 6F 6D 2C	er@tecelite.com,
0350	20 6A 6F 74 69 73 40 74	65 63 65 6C 69 74 65 2E	jotis@tecelite.
0360	63 6F 6D 2C 0D 0A 20 20	20 20 20 20 20 20 4B 69	com,.. Ki
0370	6D 20 44 61 76 69 73 20	3C 6B 64 61 76 69 73 40	m Davis <kdavis@
0380	74 65 63 65 6C 69 74 65	2E 63 6F 6D 3E 2C 20 72	tecelite.com>, r
0390	61 6D 40 74 65 63 65 6C	69 74 65 2E 63 6F 6D 2C	am@tecelite.com,
03A0	0D 0A 20 20 20 20 20 20	20 20 52 6F 62 20 52 69	.. Rob Ri
03B0	74 7A 20 3C 72 72 69 74	7A 40 74 65 63 65 6C 69	tz <rritz@teceli
03C0	74 65 2E 63 6F 6D 3E 2C	20 72 73 64 69 65 74 7A	te.com>, rsdietz
03D0	40 74 65 63 65 6C 69 74	65 2E 63 6F 6D 2C 20 73	@tecelite.com, s
03E0	6B 69 70 40 74 65 63 65	6C 69 74 65 2E 63 6F 6D	kip@tecelite.com
03F0	0D 0A 53 75 62 6A 65 63	74 3A 20 4E 65 78 74 20	..Subject: Next
0400	47 65 6E 65 72 61 74 69	6F 6E 20 50 72 6F 64 75	Generation Produ
0410	63 74 20 44 69 73 63 75	73 73 69 6F 6E 0D 0A 0D	ct Discussion...
0420	0A 0D 0A 53 75 62 6A 65	63 74 3A 20 4E 65 78 74	...Subject: Next
0430	20 47 65 6E 65 72 61 74	69 6F 6E 20 50 72 6F 64	Generation Prod
0440	75 63 74 20 44 69 73 63	75 73 73 69 6F 6E 0D 0A	uct Discussion..
0450	0D 0A 49 20 77 6F 75 6C	64 20 6C 69 6B 65 20 74	..I would like t

5 TO: 0800201310D2
FROM: 00A02475C778

2:53:51<0.198>

Pkt: 5, Len: 64/64

0000	08 00 20 13 10 D2 00 A0	24 75 C7 78 08 00 45 00\$u.x..E.
0010	00 28 B2 6C 40 00 80 06	92 1C 59 06 06 03 59 07	..(.l@.....Y...Y.

0020 FE 36 09 53 00 6E 1A 5D 8A 76 50 DA 4E A5 50 10 .6.S.n.].vP.N.P.
0030 22 38 89 41 00 00 00 00 00 00 00 BA 3C 6B D6 "8.A.....<k.

--*-*-*

6 TO: 0800201310D2
FROM: 00A02475C778

2:53:53<2.556>

Pkt: 6, Len: 66/66

0000 08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00 ..\$u.x...E.
0010 00 30 B3 6C 40 00 80 06 91 14 59 06 06 03 59 07 .0.l@....Y...Y.
0020 FE 36 09 53 00 6E 1A 5D 8A 76 50 DA 4E A5 50 18 .6.S.n.].vP.N.P.
0030 22 38 CB 6A 00 00 44 45 4C 45 20 32 0D 0A BC F6 "8.j..DELE 2....
0040 77 D9 w.

--*-*-*

7 TO: 00A02475C778
FROM: 0800201310D2

2:53:53<0.001>

Pkt: 7, Len: 91/91

0000 00 A0 24 75 C7 78 08 00 20 13 10 D2 08 00 45 00 ..\$u.x...E.
0010 00 49 16 09 00 00 3C 06 B2 5F 59 07 FE 36 59 06 .I....<..Y..6Y.
0020 06 03 00 6E 09 53 50 DA 4E A5 1A 5D 8A 7E 50 18 ...n.SP.N..].~P.
0030 10 00 3F 4C 00 00 2B 4F 4B 20 4D 65 73 73 61 67 ..?L...+OK Messag
0040 65 20 32 20 68 61 73 20 62 65 65 6E 20 64 65 6C e 2 has been del
0050 65 74 65 64 2E 0D 0A 52 E8 E2 05 eted...R...

--*-*-*

8 TO: 0800201310D2
FROM: 00A02475C778

22:53:53<0.002>

Pkt: 8, Len: 64/64

0000 08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00 ..\$u.x...E.
0010 00 2E B4 6C 40 00 80 06 90 16 59 06 06 03 59 07 ...l@....Y...Y.
0020 FE 36 09 53 00 6E 1A 5D 8A 7E 50 DA 4E C6 50 18 .6.S.n.].~P.N.P.
0030 22 17 E1 77 00 00 51 55 49 54 0D 0A 66 C7 F0 F5 "...w..QUIT..f...

--*-*-*

9 TO: 00A02475C778
FROM: 0800201310D2

2:53:53<0.029>

Pkt: 9, Len: 96/96

0000 00 A0 24 75 C7 78 08 00 20 13 10 D2 08 00 45 00 ..\$u.x...E.
0010 00 4E 16 0A 00 00 3C 06 B2 59 59 07 FE 36 59 06 .N....<..YY..6Y.
0020 06 03 00 6E 09 53 50 DA 4E C6 1A 5D 8A 84 50 18 ...n.SP.N..].~P.
0030 10 00 A0 4C 00 00 2B 4F 4B 20 50 6F 70 20 73 65 ...L...+OK Pop se
0040 72 76 65 72 20 61 74 20 73 75 70 65 72 20 73 69 rver at super si
0050 67 6E 69 6E 67 20 6F 66 66 2E 0D 0A 0D 7A D8 45 gning off....z.E

--*-*-*

10 TO: 00A02475C778
FROM: 0800201310D2

22:53:53<0.003>

Pkt: 10, Len: 64/64

0000 00 A0 24 75 C7 78 08 00 20 13 10 D2 08 00 45 00 ..\$u.x... ..E.
0010 00 28 16 0B 00 00 3C 06 B2 7E 59 07 FE 36 59 06 .(....<...Y..6Y.
0020 06 03 00 6E 09 53 50 DA 4E EC 1A 5D 8A 84 50 11 ...n.SP.N..]..P.
0030 10 00 9B 23 00 00 00 00 00 00 00 00 2B A5 6E 6A ...#.....+..nj

-

11 TO: 0800201310D2
FROM: 00A02475C778

[REDACTED]:53:53<0.000> [REDACTED]

Pkt: 11, Len: 64/64

0000 08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00\$u.x...E.
0010 00 28 B5 6C 40 00 80 06 8F 1C 59 06 06 03 59 07 .(.l@.....Y...Y.
0020 FE 36 09 53 00 6E 1A 5D 8A 84 50 DA 4E ED 50 10 .6.S.n.]..P.N.P.
0030 21 F1 89 32 00 00 00 00 00 00 00 00 BA 06 A9 CC !..2.....

-

12 TO: 0800201310D2
FROM: 00A02475C778

[REDACTED]:2:53:53<0.031> [REDACTED]

Pkt: 12, Len: 64/64

0000 08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00\$u.x...E.
0010 00 28 B6 6C 40 00 80 06 8E 1C 59 06 06 03 59 07 .(.l@.....Y...Y.
0020 FE 36 09 53 00 6E 1A 5D 8A 84 50 DA 4E ED 50 11 .6.S.n.]..P.N.P.
0030 21 F1 89 31 00 00 00 00 00 00 00 00 1C BC F9 85 !..1.....

-

13 TO: 00A02475C778
FROM: 0800201310D2

[REDACTED]:22:53:53<0.001> [REDACTED]

Pkt: 13, Len: 64/64

0000 00 A0 24 75 C7 78 08 00 20 13 10 D2 08 00 45 00 ..\$u.x... ..E.
0010 00 28 16 0C 00 00 3C 06 B2 7D 59 07 FE 36 59 06 .(....<...)Y..6Y.
0020 06 03 00 6E 09 53 50 DA 4E ED 1A 5D 8A 85 50 10 ...n.SP.N..]..P.
0030 10 00 9B 22 00 00 00 00 00 00 00 00 E0 F4 BC B0 ...".

-

14 TO: 006008C0D710
FROM: 00A076A010F2

[REDACTED]:2:53:58<4.755> [REDACTED]

Pkt: 14, Len: 64/64

0000 00 60 08 C0 D7 10 00 A0 76 A0 10 F2 08 00 45 00v.....E.
0010 00 2C 5F FE 40 00 80 06 B9 0B 59 59 18 06 59 4B .._@.....YY..YK
0020 17 18 05 B4 00 8B 01 BE 3A 7E 00 00 00 00 60 02:~.....
0030 20 00 53 E9 00 00 02 04 05 B4 20 00 D9 FB 20 4D .S..... M

-

15 TO: 00A076A010F2
FROM: 006008C0D710

[REDACTED]:22:53:58<0.001> [REDACTED]

Pkt: 15, Len: 64/64

0000 00 A0 76 A0 10 F2 00 60 08 C0 D7 10 08 00 45 00 ..v.....E.
0010 00 2C 49 7C 40 00 80 06 CF 8D 59 4B 17 18 59 59 ..I|@.....YK..YY
0020 18 06 00 8B 05 B4 1E CD 51 3B 01 BE 3A 7F 60 12Q;.....

- **Exhibit B4:** The file packets.txt that describes the nature of the packets in MFATEST.HEX.

packets.txt

***** Packet ID: 1 *****

ETHERNET=====

Destination Address: 080020-1310d2 (super)
Source Address: 006097-9d6b1d (embedded-pc)
Ethernet Type: 08-00 (IP)

IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 44 (0x2c)
Identification: 56918 (0xde56)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 32 (0x20)
Protocol: TCP(6)
Header Checksum: 7B B5
Source IP: 89.76.80.54 (embedded-pc)
Destination IP: 89.7.254.54 (super)

TCP=====

Source Port: (1427)
Destination Port: POP3(110)
Sequence Number: 16058242 (0xf50782)
Acknowledgement Number: 0
Data Offset: 6 (0x6)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 0
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 1
No More Data (FIN): 0
Window Size: 8192 (0x2000)
Checksum: 68 EE
Urgent Pointer: 0

***** Packet ID: 2 *****

ETHERNET=====

Destination Address: 006097-9d6b1d (embedded-pc)
Source Address: 080020-1310d2 (super)
Ethernet Type: 08-00 (IP)

IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 44 (0x2c)
Identification: 1630 (0x65e)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0

packets.txt

Time to Live (TTL): 60 (0x3c)
Protocol: TCP(6)
Header Checksum: 77 AE
Source IP: 89.7.254.54 (super)
Destination IP: 89.76.80.54 (embedded-pc)
TCP=====

Source Port: POP3(110)
Destination Port: (1427)
Sequence Number: 1240192000 (0x49ebd400)
Acknowledgement Number: 16058243 (0xf50783)
Data Offset: 6 (0x6)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 1
No More Data (FIN): 0
Window Size: 4096 (0x1000)
checksum: 5A F1
Urgent Pointer: 0

***** Packet ID: 3 *****

ETHERNET=====

Destination Address: 080020-1310d2 (super)
Source Address: 006097-9d6b1d (embedded-pc)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 57174 (0xdf56)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 32 (0x20)
Protocol: TCP(6)
Header Checksum: 7A B9
Source IP: 89.76.80.54 (embedded-pc)
Destination IP: 89.7.254.54 (super)
TCP=====

Source Port: (1427)
Destination Port: POP3(110)
Sequence Number: 16058243 (0xf50783)
Acknowledgement Number: 1240192001 (0x49ebd401)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 8760 (0x2238)
checksum: 60 76
Urgent Pointer: 0

packets.txt

***** Packet ID: 4 *****

ETHERNET=====

Destination Address: 006097-9d6b1d (embedded-pc)

Source Address: 080020-1310d2 (super)

Ethernet Type: 08-00 (IP)

IP=====

Version: 4

Header Length: 5 (0x5)

Type of Service: 00

TOS Precedence: Routine(0)

TOS Delay: Normal Delay(0)

TOS Throughput: Normal Throughput(0)

TOS Reliability: Normal Reliability(0)

Total Length: 120 (0x78)

Identification: 1649 (0x671)

Reserved: 0

Don't Fragment (DF): May Fragment(0)

More Fragment (MF): Last Fragment(0)

Fragment Offset: 0

Time to Live (TTL): 60 (0x3c)

Protocol: TCP(6)

Header Checksum: 77 4F

Source IP: 89.7.254.54 (super)

Destination IP: 89.76.80.54 (embedded-pc)

TCP=====

Source Port: POP3(110)

Destination Port: (1427)

Sequence Number: 1240192001 (0x49ebd401)

Acknowledgement Number: 16058243 (0xf50783)

Data Offset: 5 (0x5)

Reserved: 0

Urgent Field (URG): 0

Acknowledgement field (ACK): 1

Push Function (PSH): 1

Reset Connection (RST): 0

Synchronize Sequence (SYN): 0

No More Data (FIN): 0

Window Size: 4096 (0x1000)

Checksum: BA 88

Urgent Pointer: 0

DATA=====

Data:

```
0000 -- 2B 4F 4B 20 51 55 41 4C 43 4F +OK QUALCO
0010 -- 4D 4D 20 50 6F 70 20 73 65 72 MM Pop ser
0020 -- 76 65 72 20 64 65 72 69 76 65 ver derive
0030 -- 64 20 66 72 6F 6D 20 55 43 42 d from UCB
0040 -- 20 28 76 65 72 73 69 6F 6E 20 (version
0050 -- 32 2E 31 2E 34 2D 52 33 29 20 2.1.4-R3)
0060 -- 61 74 20 73 75 70 65 72 20 73 at super s
0070 -- 74 61 72 74 69 6E 67 2E 0D 0A tarting...
```

***** Packet ID: 5 *****

ETHERNET=====

Destination Address: 080020-1310d2 (super)

Source Address: 006097-9d6b1d (embedded-pc)

Ethernet Type: 08-00 (IP)

IP=====

Version: 4

Header Length: 5 (0x5)

Type of Service: 00

packets.txt

TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 55 (0x37)
Identification: 57430 (0xe056)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 32 (0x20)
Protocol: TCP(6)
Header Checksum: 79 AA
Source IP: 89.76.80.54 (embedded-pc)
Destination IP: 89.7.254.54 (super)
TCP=====

Source Port: (1427)
Destination Port: POP3(110)
Sequence Number: 16058243 (0xf50783)
Acknowledgement Number: 1240192081 (0x49ebd451)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 8680 (0x21e8)
Checksum: E4 02
Urgent Pointer: 0
DATA=====

Data:
0000 -- 55 53 45 52 20 6A 6D 61 69 78 USER jmaix
0010 -- 6E 65 72 0D 0A ner..

***** Packet ID: 6 *****

ETHERNET=====

Destination Address: 006097-9d6b1d (embedded-pc)
Source Address: 080020-1310d2 (super)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 77 (0x4d)
Identification: 1650 (0x672)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 60 (0x3c)
Protocol: TCP(6)
Header Checksum: 77 79
Source IP: 89.7.254.54 (super)
Destination IP: 89.76.80.54 (embedded-pc)
TCP=====

Source Port: POP3(110)

packets.txt

Destination Port: (1427)
Sequence Number: 1240192081 (0x49ebd451)
Acknowledgement Number: 16058258 (0xf50792)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 4096 (0x1000)
Checksum: C1 E2
Urgent Pointer: 0

DATA=====

Data:

0000 -- 2B 4F 4B 20 50 61 73 73 77 6F +OK Passwo
0010 -- 72 64 20 72 65 71 75 69 72 65 rd require
0020 -- 64 20 66 6F 72 20 6A 6D 61 69 d for jmai
0030 -- 78 6E 65 72 2E 0D 0A xner...

***** Packet ID: 7 *****

ETHERNET=====

Destination Address: 080020-1310d2 (super)
Source Address: 006097-9d6b1d (embedded-pc)
Ethernet Type: 08-00 (IP)

IP=====

Version: 4

Header Length: 5 (0x5)

Type of Service: 00

TOS Precedence: Routine(0)

TOS Delay: Normal Delay(0)

TOS Throughput: Normal Throughput(0)

TOS Reliability: Normal Reliability(0)

Total Length: 55 (0x37)

Identification: 57686 (0xe156)

Reserved: 0

Don't Fragment (DF): Don't Fragment(1)

More Fragment (MF): Last Fragment(0)

Fragment Offset: 0

Time to Live (TTL): 32 (0x20)

Protocol: TCP(6)

Header Checksum: 78 AA

Source IP: 89.76.80.54 (embedded-pc)

Destination IP: 89.7.254.54 (super)

TCP=====

Source Port: (1427)

Destination Port: POP3(110)

Sequence Number: 16058258 (0xf50792)

Acknowledgement Number: 1240192118 (0x49ebd476)

Data Offset: 5 (0x5)

Reserved: 0

Urgent Field (URG): 0

Acknowledgement field (ACK): 1

Push Function (PSH): 1

Reset Connection (RST): 0

Synchronize Sequence (SYN): 0

No More Data (FIN): 0

Window Size: 8643 (0x21c3)

Checksum: DB 04

Urgent Pointer: 0

DATA=====

packets.txt

Data:

0000 -- 50 41 53 53 20 6A 6D 61 69 78 PASS jmaix
0010 -- 6E 65 72 0D 0A ner..

***** Packet ID: 8 *****

ETHERNET=====

Destination Address: 006097-9d6b1d (embedded-pc)
Source Address: 080020-1310d2 (super)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 1651 (0x673)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 60 (0x3c)
Protocol: TCP(6)
Header Checksum: 77 9D
Source IP: 89.7.254.54 (super)
Destination IP: 89.76.80.54 (embedded-pc)
TCP=====

Source Port: POP3(110)
Destination Port: (1427)
Sequence Number: 1240192118 (0x49ebd476)
Acknowledgement Number: 16058273 (0xf507a1)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 4096 (0x1000)
Checksum: 72 1B
Urgent Pointer: 0

***** Packet ID: 9 *****

ETHERNET=====

Destination Address: 006097-9d6b1d (embedded-pc)
Source Address: 080020-1310d2 (super)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 83 (0x53)
Identification: 1654 (0x676)
Reserved: 0

packets.txt

Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 60 (0x3c)
Protocol: TCP(6)
Header Checksum: 77 6F
Source IP: 89.7.254.54 (super)
Destination IP: 89.76.80.54 (embedded-pc)
TCP=====

Source Port: POP3(110)
Destination Port: (1427)
Sequence Number: 1240192118 (0x49ebd476)
Acknowledgement Number: 16058273 (0xf507a1)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 4096 (0x1000)
Checksum: 40 BC
Urgent Pointer: 0
DATA=====

Data:

0000	--	2B 4F 4B 20 6A 6D 61 69 78 6E	+OK jmaixn
0010	--	65 72 20 68 61 73 20 30 20 6D	er has 0 m
0020	--	65 73 73 61 67 65 28 73 29 20	essage(s)
0030	--	28 30 20 6F 63 74 65 74 73 29	(0 octets)
0040	--	2E 0D 0A	...

***** Packet ID: 10 *****

ETHERNET=====

Destination Address: 080020-1310d2 (super)
Source Address: 006097-9d6b1d (embedded-pc)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 46 (0x2e)
Identification: 57942 (0xe256)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 32 (0x20)
Protocol: TCP(6)
Header Checksum: 77 B3
Source IP: 89.76.80.54 (embedded-pc)
Destination IP: 89.7.254.54 (super)
TCP=====

Source Port: (1427)
Destination Port: POP3(110)
Sequence Number: 16058273 (0xf507a1)
Acknowledgement Number: 1240192161 (0x49ebd4a1)
Data Offset: 5 (0x5)

packets.txt

Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 8600 (0x2198)
Checksum: BE 97
Urgent Pointer: 0
DATA=====

Data: 53 54 41 54 0D 0A STAT..

***** Packet ID: 11 *****

ETHERNET=====

Destination Address: 006097-9d6b1d (embedded-pc)
Source Address: 080020-1310d2 (super)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 49 (0x31)
Identification: 1655 (0x677)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 60 (0x3c)
Protocol: TCP(6)
Header checksum: 77 90
Source IP: 89.7.254.54 (super)
Destination IP: 89.76.80.54 (embedded-pc)
TCP=====

Source Port: POP3(110)
Destination Port: (1427)
Sequence Number: 1240192161 (0x49ebd4a1)
Acknowledgement Number: 16058279 (0xf507a7)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 4096 (0x1000)
Checksum: 91 3c
Urgent Pointer: 0
DATA=====

Data: 2B 4F 4B 20 30 20 30 0D 0A +OK 0 0..

***** Packet ID: 12 *****

ETHERNET=====

Destination Address: 080020-1310d2 (super)
Source Address: 006097-9d6b1d (embedded-pc)
Ethernet Type: 08-00 (IP)

packets.txt

IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 46 (0x2e)
Identification: 58198 (0xe356)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 32 (0x20)
Protocol: TCP(6)
Header Checksum: 76 B3
Source IP: 89.76.80.54 (embedded-pc)
Destination IP: 89.7.254.54 (super)

TCP=====

Source Port: (1427)
Destination Port: POP3(110)
Sequence Number: 16058279 (0xf507a7)
Acknowledgement Number: 1240192170 (0x49ebd4aa)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 8591 (0x218f)
Checksum: B8 90
Urgent Pointer: 0

DATA=====

Data: 51 55 49 54 0D 0A QUIT..

***** Packet ID: 13 *****

ETHERNET=====

Destination Address: 006097-9d6b1d (embedded-pc)
Source Address: 080020-1310d2 (super)
Ethernet Type: 08-00 (IP)

IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 78 (0x4e)
Identification: 1656 (0x678)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 60 (0x3c)
Protocol: TCP(6)
Header Checksum: 77 72
Source IP: 89.7.254.54 (super)
Destination IP: 89.76.80.54 (embedded-pc)

packets.txt

```
TCP=====
Source Port: POP3(110)
Destination Port: (1427)
Sequence Number: 1240192170 (0x49ebd4aa)
Acknowledgement Number: 16058285 (0xf507ad)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 4096 (0x1000)
Checksum: 76 DD
Urgent Pointer: 0
DATA=====
Data:
0000 -- 2B 4F 4B 20 50 6F 70 20 73 65 +OK Pop se
0010 -- 72 76 65 72 20 61 74 20 73 75 rver at su
0020 -- 70 65 72 20 73 69 67 6E 69 6E per signin
0030 -- 67 20 6F 66 66 2E 0D 0A g off...
```

***** Packet ID: 14 *****

```
ETHERNET=====
Destination Address: 080020-1310d2 (super)
Source Address: 006097-9d6b1d (embedded-pc)
Ethernet Type: 08-00 (IP)
IP=====
Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 58454 (0xe456)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 32 (0x20)
Protocol: TCP(6)
Header Checksum: 75 B9
Source IP: 89.76.80.54 (embedded-pc)
Destination IP: 89.7.254.54 (super)
TCP=====
Source Port: (1427)
Destination Port: POP3(110)
Sequence Number: 16058285 (0xf507ad)
Acknowledgement Number: 1240192208 (0x49ebd4d0)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 1
Window Size: 8553 (0x2169)
Checksum: 60 4B
```

Urgent Pointer: 0

******* Packet ID: 15 *******

```
ETHERNET=====
Destination Address: 006097-9d6b1d (embedded-pc)
Source Address: 080020-1310d2 (super)
Ethernet Type: 08-00 (IP)
IP=====
Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 1657 (0x679)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 60 (0x3c)
Protocol: TCP(6)
Header Checksum: 77 97
Source IP: 89.7.254.54 (super)
Destination IP: 89.76.80.54 (embedded-pc)
TCP=====
Source Port: POP3(110)
Destination Port: (1427)
Sequence Number: 1240192208 (0x49ebd4d0)
Acknowledgement Number: 16058286 (0xf507ae)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 4096 (0x1000)
Checksum: 71 B4
Urgent Pointer: 0
```

******* Packet ID: 16 *******

```
ETHERNET=====
Destination Address: 006097-9d6b1d (embedded-pc)
Source Address: 080020-1310d2 (super)
Ethernet Type: 08-00 (IP)
IP=====
Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 1658 (0x67a)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
```

packets.txt

Fragment Offset: 0
Time to Live (TTL): 60 (0x3c)
Protocol: TCP(6)
Header Checksum: 77 96
Source IP: 89.7.254.54 (super)
Destination IP: 89.76.80.54 (embedded-pc)
TCP=====

Source Port: POP3(110)
Destination Port: (1427)
Sequence Number: 1240192208 (0x49ebd4d0)
Acknowledgement Number: 16058286 (0xf507ae)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 1
Window Size: 4096 (0x1000)
Checksum: 71 B3
Urgent Pointer: 0

***** Packet ID: 17 *****

ETHERNET=====

Destination Address: 080020-1310d2 (super)
Source Address: 006097-9d6b1d (embedded-pc)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 58710 (0xe556)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 32 (0x20)
Protocol: TCP(6)
Header Checksum: 74 B9
Source IP: 89.76.80.54 (embedded-pc)
Destination IP: 89.7.254.54 (super)
TCP=====

Source Port: (1427)
Destination Port: POP3(110)
Sequence Number: 16058286 (0xf507ae)
Acknowledgement Number: 1240192209 (0x49ebd4d1)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 8553 (0x2169)
Checksum: 60 4A

Urgent Pointer: 0

***** Packet ID: 18 *****

ETHERNET=====

Destination Address: 080020-0dddf9 (c3po)
Source Address: 0000a3-b0022a (TecElite-N.b0022a)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 37459 (0x9253)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 61 (0x3d)
Protocol: TCP(6)
Header Checksum: 15 3D
Source IP: 192.190.175.254 (ftp)
Destination IP: 89.111.12.20 (c3po)
TCP=====

Source Port: TELNET(23)
Destination Port: (32779)
Sequence Number: 3652221321 (0xd9b07989)
Acknowledgement Number: 4022713487 (0xefc5bc8f)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 32736 (0x7fe0)
Checksum: DA 01
Urgent Pointer: 0

***** Packet ID: 19 *****

ETHERNET=====

Destination Address: 0000a3-b0022a (TecElite-N.b0022a)
Source Address: 080020-0dddf9 (c3po)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 21585 (0x5451)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)

packets.txt

Fragment Offset: 0
Time to Live (TTL): 255 (0xff)
Protocol: TCP(6)
Header Checksum: 51 3E
Source IP: 89.111.12.20 (c3po)
Destination IP: 192.190.175.254 (ftp)
TCP=====

Source Port: (32779)
Destination Port: TELNET(23)
Sequence Number: 4022713487 (0xefc5bc8f)
Acknowledgement Number: 3652221322 (0xd9b0798a)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 8760 (0x2238)
Checksum: 37 A9
Urgent Pointer: 0

***** Packet ID: 20 *****

ETHERNET=====

Destination Address: aa0004-000104 (DEC.000104)
Source Address: 0000e8-061f15 (smtp1ink)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 44 (0x2c)
Identification: 3736 (0xe98)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 64 (0x40)
Protocol: TCP(6)
Header Checksum: 9B 0c
Source IP: 89.7.7.100 (smtp1ink)
Destination IP: 192.190.175.254 (ftp)
TCP=====

Source Port: (11348)
Destination Port: SMTP(25)
Sequence Number: 104679649 (0x63d48e1)
Acknowledgement Number: 1 (0x1)
Data Offset: 6 (0x6)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 0
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 1
No More Data (FIN): 0
Window Size: 2048 (0x800)
Checksum: 47 0E

Urgent Pointer: 0

***** Packet ID: 21 *****

```
ETHERNET=====
Destination Address: 0000e8-061f15 (smtp1ink)
Source Address: 0000a3-b0022a (TecElite-N.b0022a)
Ethernet Type: 08-00 (IP)
IP=====
Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 44 (0x2c)
Identification: 37475 (0x9263)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 62 (0x3e)
Protocol: TCP(6)
Header Checksum: 19 41
Source IP: 192.190.175.254 (ftp)
Destination IP: 89.7.7.100 (smtp1ink)
TCP=====
Source Port: SMTP(25)
Destination Port: (11348)
Sequence Number: 3878102034 (0xe7272412)
Acknowledgement Number: 104679650 (0x63d48e2)
Data Offset: 6 (0x6)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 1
No More Data (FIN): 0
Window Size: 32736 (0x7fe0)
checksum: c3 e3
Urgent Pointer: 0
```

***** Packet ID: 22 *****

```
ETHERNET=====
Destination Address: aa0004-000104 (DEC.000104)
Source Address: 0000e8-061f15 (smtp1ink)
Ethernet Type: 08-00 (IP)
IP=====
Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 3737 (0xe99)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
```

packets.txt

Fragment Offset: 0
Time to Live (TTL): 64 (0x40)
Protocol: TCP(6)
Header Checksum: 9B 0F
Source IP: 89.7.7.100 (smtp1ink)
Destination IP: 192.190.175.254 (ftp)
TCP=====

Source Port: (11348)
Destination Port: SMTP(25)
Sequence Number: 104679650 (0x63d48e2)
Acknowledgement Number: 3878102035 (0xe7272413)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 2048 (0x800)
Checksum: 4F E5
Urgent Pointer: 0

***** Packet ID: 23 *****

ETHERNET=====

Destination Address: 0000e8-061f15 (smtp1ink)
Source Address: 0000a3-b0022a (TecElite-N.b0022a)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 44 (0x2c)
Identification: 37476 (0x9264)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 62 (0x3e)
Protocol: TCP(6)
Header Checksum: 19 40
Source IP: 192.190.175.254 (ftp)
Destination IP: 89.7.7.100 (smtp1ink)
TCP=====

Source Port: (12998)
Destination Port: (113)
Sequence Number: 2356842160 (0x8c7a8eb0)
Acknowledgement Number: 0
Data Offset: 6 (0x6)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 0
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 1
No More Data (FIN): 0
Window Size: 512 (0x200)
Checksum: 76 9C

Urgent Pointer: 0

***** Packet ID: 24 *****

```
ETHERNET=====
Destination Address: aa0004-000104 (DEC.000104)
Source Address: 0000e8-061f15 (smtp1ink)
Ethernet Type: 08-00 (IP)
IP=====
Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 3738 (0xe9a)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 64 (0x40)
Protocol: TCP(6)
Header Checksum: 9B 0E
Source IP: 89.7.7.100 (smtp1ink)
Destination IP: 192.190.175.254 (ftp)
TCP=====
Source Port: (113)
Destination Port: (12998)
Sequence Number: 0
Acknowledgement Number: 2356842161 (0x8c7a8eb1)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 1
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 0
Checksum: 90 3D
Urgent Pointer: 0
```

***** Packet ID: 25 *****

```
ETHERNET=====
Destination Address: 0000e8-061f15 (smtp1ink)
Source Address: 0000a3-b0022a (TecElite-N.b0022a)
Ethernet Type: 08-00 (IP)
IP=====
Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 125 (0x7d)
Identification: 37477 (0x9265)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
```

packets.txt

Fragment Offset: 0
Time to Live (TTL): 62 (0x3e)
Protocol: TCP(6)
Header Checksum: D8 ED
Source IP: 192.190.175.254 (ftp)
Destination IP: 89.7.7.100 (smtpLink)
TCP=====

Source Port: SMTP(25)
Destination Port: (11348)
Sequence Number: 3878102035 (0xe7272413)
Acknowledgement Number: 104679650 (0x63d48e2)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 32736 (0x7fe0)
Checksum: FC 9F
Urgent Pointer: 0
DATA=====

Data:

0000	--	32	32	30	20	6E	61	74	61	64	6D	220 natadm
0010	--	2E	74	65	63	65	6C	69	74	65	2E	.tecelite.
0020	--	63	6F	6D	20	45	53	4D	54	50	20	com ESMTP
0030	--	53	65	6E	64	6D	61	69	6C	20	38	sendmail 8
0040	--	2E	38	2E	37	2F	38	2E	38	2E	37	.8.7/8.8.7
0050	--	3B	20	54	68	75	2C	20	31	30	20	; Thu, 10
0060	--	53	65	70	20	31	39	39	38	20	31	Sep 1998 1
0070	--	37	3A	32	38	3A	31	30	20	2D	30	7:28:10 -0
0080	--	37	30	30	0D	0A						700..

***** Packet ID: 26 *****

ETHERNET=====

Destination Address: aa0004-000104 (DEC.000104)
Source Address: 0000e8-061f15 (smtpLink)
Ethernet Type: 08-00 (IP)
IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 68 (0x44)
Identification: 3739 (0xe9b)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 64 (0x40)
Protocol: TCP(6)
Header Checksum: 9A F1
Source IP: 89.7.7.100 (smtpLink)
Destination IP: 192.190.175.254 (ftp)
TCP=====

Source Port: (11348)
Destination Port: SMTP(25)
Sequence Number: 104679650 (0x63d48e2)

packets.txt
Acknowledgement Number: 3878102120 (0xe7272468)

Data Offset: 5 (0x5)

Reserved: 0

Urgent Field (URG): 0

Acknowledgement field (ACK): 1

Push Function (PSH): 1

Reset Connection (RST): 0

Synchronize Sequence (SYN): 0

No More Data (FIN): 0

Window Size: 1963 (0x7ab)

checksum: 84 c7

Urgent Pointer: 0

DATA=====

Data:

0000 -- 48 45 4C 4F 20 73 6D 74 70 6C HELO smtp1

0010 -- 69 6E 6B 2E 74 65 63 65 6C 69 ink.teceli

0020 -- 74 65 2E 63 6F 6D 0D 0A te.com..

***** Packet ID: 27 *****

ETHERNET=====

Destination Address: 0000e8-061f15 (smtp1ink)

Source Address: 0000a3-b0022a (TecElite-N.b0022a)

Ethernet Type: 08-00 (IP)

IP=====

Version: 4

Header Length: 5 (0x5)

Type of Service: 00

TOS Precedence: Routine(0)

TOS Delay: Normal Delay(0)

TOS Throughput: Normal Throughput(0)

TOS Reliability: Normal Reliability(0)

Total Length: 114 (0x72)

Identification: 37478 (0x9266)

Reserved: 0

Don't Fragment (DF): Don't Fragment(1)

More Fragment (MF): Last Fragment(0)

Fragment Offset: 0

Time to Live (TTL): 62 (0x3e)

Protocol: TCP(6)

Header Checksum: D8 F7

Source IP: 192.190.175.254 (ftp)

Destination IP: 89.7.7.100 (smtp1ink)

TCP=====

Source Port: SMTP(25)

Destination Port: (11348)

Sequence Number: 3878102120 (0xe7272468)

Acknowledgement Number: 104679678 (0x63d48fe)

Data Offset: 5 (0x5)

Reserved: 0

Urgent Field (URG): 0

Acknowledgement field (ACK): 1

Push Function (PSH): 1

Reset Connection (RST): 0

Synchronize Sequence (SYN): 0

No More Data (FIN): 0

Window Size: 32736 (0x7fe0)

Checksum: EA FB

Urgent Pointer: 0

DATA=====

Data:

0000 -- 32 35 30 20 6E 61 74 61 64 6D 250 natadm

0010 -- 2E 74 65 63 65 6C 69 74 65 2E .tecelite.

```

                                packets.txt
0020 -- 63 6F 6D 20 48 65 6C 6C 6F 20 com Hello
0030 -- 73 6D 74 70 6C 69 6E 6B 20 5B smtpLink [
0040 -- 38 39 2E 37 2E 37 2E 31 30 30 89.7.7.100
0050 -- 5D 2C 20 70 6C 65 61 73 65 64 ], please
0060 -- 20 74 6F 20 6D 65 65 74 20 79 to meet y
0070 -- 6F 75 0D 0A ou..

```

***** Packet ID: 28 *****

```

ETHERNET=====
Destination Address: aa0004-000104 (DEC.000104)
Source Address: 0000e8-061f15 (smtpLink)
Ethernet Type: 08-00 (IP)
IP=====
Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 88 (0x58)
Identification: 3740 (0xe9c)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 64 (0x40)
Protocol: TCP(6)
Header Checksum: 9A DC
Source IP: 89.7.7.100 (smtpLink)
Destination IP: 192.190.175.254 (ftp)
TCP=====
Source Port: (11348)
Destination Port: SMTP(25)
Sequence Number: 104679678 (0x63d48fe)
Acknowledgement Number: 3878102194 (0xe72724b2)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 1889 (0x761)
Checksum: 6B B6
Urgent Pointer: 0
DATA=====
Data:
0000 -- 4D 41 49 4C 20 46 52 4F 4D 3A MAIL FROM:
0010 -- 3C 44 6F 75 67 20 46 65 6C 64 <Doug.Feld
0020 -- 65 72 20 3C 64 66 65 6C 64 65 er <dfelde
0030 -- 72 40 74 65 63 65 6C 69 74 65 r@tecelite
0040 -- 2E 63 6F 6D 3E 3E 0D 0A .com>>..

```

***** Packet ID: 29 *****

```

ETHERNET=====
Destination Address: 0000e8-061f15 (smtpLink)
Source Address: 0000a3-b0022a (TecElite-N.b0022a)
Ethernet Type: 08-00 (IP)
IP=====

```

```

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 37481 (0x9269)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 62 (0x3e)
Protocol: TCP(6)
Header Checksum: D9 3E
Source IP: 192.190.175.254 (ftp)
Destination IP: 89.7.7.100 (smtpLink)
TCP=====
Source Port: SMTP(25)
Destination Port: (11348)
Sequence Number: 3878102194 (0xe72724b2)
Acknowledgement Number: 104679726 (0x63d492e)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 32736 (0x7fe0)
Checksum: D7 19
Urgent Pointer: 0

```

***** Packet ID: 30 *****

```

ETHERNET=====
Destination Address: 0000e8-061f15 (smtpLink)
Source Address: 0000a3-b0022a (TecElite-N.b0022a)
Ethernet Type: 08-00 (IP)
IP=====
Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 95 (0x5f)
Identification: 37482 (0x926a)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 62 (0x3e)
Protocol: TCP(6)
Header Checksum: D9 06
Source IP: 192.190.175.254 (ftp)
Destination IP: 89.7.7.100 (smtpLink)
TCP=====
Source Port: SMTP(25)
Destination Port: (11348)

```


packets.txt

```

Sequence Number: 3878102194 (0xe72724b2)
Acknowledgement Number: 104679726 (0x63d492e)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 32736 (0x7fe0)
Checksum: DB 0A
Urgent Pointer: 0

```

```

DATA=====
Data:
0000 -- 32 35 30 20 3C 44 6F 75 67 20 250 <Doug
0010 -- 46 65 6C 64 65 72 20 3C 64 66 Felder <df
0020 -- 65 6C 64 65 72 40 74 65 63 65 elder@tece
0030 -- 6C 69 74 65 2E 63 6F 6D 3E 3E lite.com>>
0040 -- 2E 2E 2E 20 53 65 6E 64 65 72 ... Sender
0050 -- 20 6F 6B 0D 0A ok..

```

***** Packet ID: 31 *****

```

ETHERNET=====
Destination Address: aa0004-000104 (DEC.000104)
Source Address: 0000e8-061f15 (smtp1ink)
Ethernet Type: 08-00 (IP)
IP=====
Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 70 (0x46)
Identification: 3741 (0xe9d)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 64 (0x40)
Protocol: TCP(6)
Header Checksum: 9A ED
Source IP: 89.7.7.100 (smtp1ink)
Destination IP: 192.190.175.254 (ftp)
TCP=====
Source Port: (11348)
Destination Port: SMTP(25)
Sequence Number: 104679726 (0x63d492e)
Acknowledgement Number: 3878102249 (0xe72724e9)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 1834 (0x72a)
Checksum: 43 E8
Urgent Pointer: 0

```

packets.txt

DATA=====

Data:

0000 -- 52 43 50 54 20 54 4F 3A 3C 6B RCPT TO:<k
0010 -- 61 68 6D 69 6E 2E 74 65 68 40 ahmin.teh@
0020 -- 61 6D 64 2E 63 6F 6D 3E 0D 0A amd.com>..

***** Packet ID: 32 *****

ETHERNET=====

Destination Address: 0000e8-061f15 (smtp1ink)
Source Address: 0000a3-b0022a (TecElite-N.b0022a)
Ethernet Type: 08-00 (IP)

IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 40 (0x28)
Identification: 37485 (0x926d)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 62 (0x3e)
Protocol: TCP(6)
Header Checksum: D9 3A
Source IP: 192.190.175.254 (ftp)
Destination IP: 89.7.7.100 (smtp1ink)
TCP=====

Source Port: SMTP(25)
Destination Port: (11348)
Sequence Number: 3878102249 (0xe72724e9)
Acknowledgement Number: 104679756 (0x63d494c)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 0
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 32736 (0x7fe0)
Checksum: D6 C4
Urgent Pointer: 0

***** Packet ID: 33 *****

ETHERNET=====

Destination Address: 0000e8-061f15 (smtp1ink)
Source Address: 0000a3-b0022a (TecElite-N.b0022a)
Ethernet Type: 08-00 (IP)

IP=====

Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 82 (0x52)

```

Identification: 37493 (0x9275)
Reserved: 0
Don't Fragment (DF): Don't Fragment(1)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 62 (0x3e)
Protocol: TCP(6)
Header Checksum: D9 08
Source IP: 192.190.175.254 (ftp)
Destination IP: 89.7.7.100 (smtp)link)
TCP=====
Source Port: SMTP(25)
Destination Port: (11348)
Sequence Number: 3878102249 (0xe72724e9)
Acknowledgement Number: 104679756 (0x63d494c)
Data Offset: 5 (0x5)
Reserved: 0
Urgent Field (URG): 0
Acknowledgement field (ACK): 1
Push Function (PSH): 1
Reset Connection (RST): 0
Synchronize Sequence (SYN): 0
No More Data (FIN): 0
Window Size: 32736 (0x7fe0)
Checksum: AF 57
Urgent Pointer: 0
DATA=====
Data:
0000 -- 32 35 30 20 3C 6B 61 68 6D 69 250 <kahmi
0010 -- 6E 2E 74 65 68 40 61 6D 64 2E n.teh@amd.
0020 -- 63 6F 6D 3E 2E 2E 2E 20 52 65 com>... Re
0030 -- 63 69 70 69 65 6E 74 20 6F 6B cipient ok
0040 -- 0D 0A ..

```

***** Packet ID: 34 *****

```

ETHERNET=====
Destination Address: aa0004-000104 (DEC.000104)
Source Address: 0000e8-061f15 (smtp)link)
Ethernet Type: 08-00 (IP)
IP=====
Version: 4
Header Length: 5 (0x5)
Type of Service: 00
TOS Precedence: Routine(0)
TOS Delay: Normal Delay(0)
TOS Throughput: Normal Throughput(0)
TOS Reliability: Normal Reliability(0)
Total Length: 47 (0x2f)
Identification: 3742 (0xe9e)
Reserved: 0
Don't Fragment (DF): May Fragment(0)
More Fragment (MF): Last Fragment(0)
Fragment Offset: 0
Time to Live (TTL): 64 (0x40)
Protocol: TCP(6)
Header Checksum: 9B 03
Source IP: 89.7.7.100 (smtp)link)
Destination IP: 192.190.175.254 (ftp)
TCP=====
Source Port: (11348)
Destination Port: SMTP(25)
Sequence Number: 104679756 (0x63d494c)

```

packets.txt
Acknowledgement Number: 3878102291 (0xe7272513)

Data Offset: 5 (0x5)

Reserved: 0

Urgent Field (URG): 0

Acknowledgement field (ACK): 1

Push Function (PSH): 1

Reset Connection (RST): 0

Synchronize Sequence (SYN): 0

No More Data (FIN): 0

Window Size: 1792 (0x700)

Checksum: 8C DC

Urgent Pointer: 0

DATA=====

Data: 44 41 54 41 20 0D 0A DATA ..

***** Packet ID: 35 *****

ETHERNET=====

Destination Address: 0000e8-061f15 (smtp1ink)

Source Address: 0000a3-b0022a (TecElite-N.b0022a)

Ethernet Type: 08-00 (IP)

IP=====

Version: 4

Header Length: 5 (0x5)

Type of Service: 00

TOS Precedence: Routine(0)

TOS Delay: Normal Delay(0)

TOS Throughput: Normal Throughput(0)

TOS Reliability: Normal Reliability(0)

Total Length: 90 (0x5a)

Identification: 37494 (0x9276)

Reserved: 0

Don't Fragment (DF): Don't Fragment(1)

More Fragment (MF): Last Fragment(0)

Fragment Offset: 0

Time to Live (TTL): 62 (0x3e)

Protocol: TCP(6)

Header Checksum: D8 FF

Source IP: 192.190.175.254 (ftp)

Destination IP: 89.7.7.100 (smtp1ink)

TCP=====

Source Port: SMTP(25)

Destination Port: (11348)

Sequence Number: 3878102291 (0xe7272513)

Acknowledgement Number: 104679763 (0x63d4953)

Data Offset: 5 (0x5)

Reserved: 0

Urgent Field (URG): 0

Acknowledgement field (ACK): 1

Push Function (PSH): 1

Reset Connection (RST): 0

Synchronize Sequence (SYN): 0

N

- **Exhibit B5:** The contents of files mfaptpkt.txt and mfaptpkt2.txt that are files that contain the elements that were extracted by the parsing/extracting of

mfaptpkt.txt

42

08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00
00 30 B0 6C 40 00 80 06 94 14 59 06 06 03 59 07
FE 36 09 53 00 6E 1A 5D 8A 6E 50 DA 49 60 50 18
1D 4B BF 97 00 00 52 45 54 52 20 32 0D 0A FD 6E
9D F5

-

4B

00 A0 24 75 C7 78 08 00 20 13 10 D2 08 00 45 00
00 39 16 03 00 00 3C 06 B2 75 59 07 FE 36 59 06
06 03 00 6E 09 53 50 DA 49 60 1A 5D 8A 76 50 18
10 00 5D 6C 00 00 2B 4F 4B 20 31 33 35 31 20 6F
63 74 65 74 73 0D 0A CA E0 6A B1

-

40

08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00
00 28 B1 6C 40 00 80 06 93 1C 59 06 06 03 59 07
FE 36 09 53 00 6E 1A 5D 8A 76 50 DA 49 71 50 10
1D 3A 93 73 00 00 00 00 00 00 00 02 03 21 C2

-

460

00 A0 24 75 C7 78 08 00 20 13 10 D2 08 00 45 00
05 5C 16 05 00 00 3C 06 AD 50 59 07 FE 36 59 06
06 03 00 6E 09 53 50 DA 49 71 1A 5D 8A 76 50 18
10 00 53 11 00 00 52 65 74 75 72 6E 2D 50 61 74
68 3A 20 3C 6A 6D 65 74 7A 67 65 72 40 74 65 63
65 6C 69 74 65 2E 63 6F 6D 3E 0D 0A 52 65 63 65
69 76 65 64 3A 20 66 72 6F 6D 20 6E 61 74 61 64
6D 2E 74 65 63 65 6C 69 74 65 2E 63 6F 6D 20 62
79 20 73 75 70 65 72 2E 74 65 63 65 6C 69 74 65
2E 63 6F 6D 20 28 34 2E 31 2F 53 4D 49 2D 34 2E
31 29 0D 0A 09 69 64 20 41 41 32 38 34 30 38 3B
20 54 68 75 2C 20 31 30 20 53 65 70 20 39 38 20
31 37 3A 33 37 3A 33 37 20 50 44 54 0D 0A 52 65
63 65 69 76 65 64 3A 20 66 72 6F 6D 20 73 6D 74
70 6C 69 6E 6B 2E 74 65 63 65 6C 69 74 65 2E 63
6F 6D 20 28 73 6D 74 70 6C 69 6E 6B 20 5B 38 39
2E 37 2E 37 2E 31 30 30 5D 29 0D 0A 09 62 79 20
6E 61 74 61 64 6D 2E 74 65 63 65 6C 69 74 65 2E
63 6F 6D 20 28 38 2E 38 2E 37 2F 38 2E 38 2E 37
29 20 77 69 74 68 20 53 4D 54 50 20 69 64 20 52
41 41 31 37 32 34 35 3B 0D 0A 09 54 68 75 2C 20
31 30 20 53 65 70 20 31 39 39 38 20 31 37 3A 33
39 3A 30 34 20 2D 30 37 30 30 0D 0A 52 65 63 65
69 76 65 64 3A 20 66 72 6F 6D 20 63 63 3A 4D 61
69 6C 20 62 79 20 73 6D 74 70 6C 69 6E 6B 2E 74
65 63 65 6C 69 74 65 2E 63 6F 6D 0D 0A 09 69 64
20 41 41 39 30 35 34 37 34 38 32 39 20 54 68 75
2C 20 31 30 20 53 65 70 20 39 38 20 31 37 3A 34
37 3A 30 39 20 50 44 54 0D 0A 44 61 74 65 3A 20
54 68 75 2C 20 31 30 20 53 65 70 20 39 38 20 31
37 3A 34 37 3A 30 39 20 50 44 54 0D 0A 46 72 6F
6D 3A 20 4A 6F 68 6E 20 4D 65 74 7A 67 65 72 20
3C 6A 6D 65 74 7A 67 65 72 40 74 65 63 65 6C 69
74 65 2E 63 6F 6D 3E 0D 0A 45 6E 63 6F 64 69 6E
67 3A 20 33 32 34 20 54 65 78 74 0D 0A 4D 65 73
73 61 67 65 2D 49 64 3A 20 3C 39 38 30 38 31 30
39 30 35 34 2E 41 41 39 30 35 34 37 34 38 32 39
40 73 6D 74 70 6C 69 6E 6B 2E 74 65 63 65 6C 69
74 65 2E 63 6F 6D 3E 0D 0A 54 6F 3A 20 62 6C 65
61 76 79 40 74 65 63 65 6C 69 74 65 2E 63 6F 6D
2C 20 61 63 68 61 64 64 61 40 74 65 63 65 6C 69
74 65 2E 63 6F 6D 2C 20 64 61 76 65 63 40 74 65

mfaptpkt.txt

```
63 65 6C 69 74 65 2E 63 6F 6D 2C 0D 0A 20 20 20
20 20 20 20 20 44 61 76 69 64 20 4C 75 6F 20 3C
64 6C 75 6F 40 74 65 63 65 6C 69 74 65 2E 63 6F
6D 3E 2C 20 6C 6F 77 64 65 72 40 74 65 63 65 6C
69 74 65 2E 63 6F 6D 2C 0D 0A 20 20 20 20 20 20
20 20 65 77 68 65 65 6C 65 72 40 74 65 63 65 6C
69 74 65 2E 63 6F 6D 2C 20 66 6E 6F 6F 6E 40 74
65 63 65 6C 69 74 65 2E 63 6F 6D 2C 20 66 72 65
64 6D 40 74 65 63 65 6C 69 74 65 2E 63 6F 6D 2C
0D 0A 20 20 20 20 20 20 20 20 20 6A 6D 61 69 78 6E
65 72 40 74 65 63 65 6C 69 74 65 2E 63 6F 6D 2C
20 6A 6F 74 69 73 40 74 65 63 65 6C 69 74 65 2E
63 6F 6D 2C 0D 0A 20 20 20 20 20 20 20 4B 69
6D 20 44 61 76 69 73 20 3C 6B 64 61 76 69 73 40
74 65 63 65 6C 69 74 65 2E 63 6F 6D 3E 2C 20 72
61 6D 40 74 65 63 65 6C 69 74 65 2E 63 6F 6D 2C
0D 0A 20 20 20 20 20 20 20 20 20 52 6F 62 20 52 69
74 7A 20 3C 72 72 69 74 7A 40 74 65 63 65 6C 69
74 65 2E 63 6F 6D 3E 2C 20 72 73 64 69 65 74 7A
40 74 65 63 65 6C 69 74 65 2E 63 6F 6D 2C 20 73
6B 69 70 40 74 65 63 65 6C 69 74 65 2E 63 6F 6D
0D 0A 53 75 62 6A 65 63 74 3A 20 4E 65 78 74 20
47 65 6E 65 72 61 74 69 6F 6E 20 50 72 6F 64 75
63 74 20 44 69 73 63 75 73 73 69 6F 6E 0D 0A 0D
0A 0D 0A 53 75 62 6A 65 63 74 3A 20 4E 65 78 74
20 47 65 6E 65 72 61 74 69 6F 6E 20 50 72 6F 64
75 63 74 20 44 69 73 63 75 73 73 69 6F 6E 0D 0A
0D 0A 49 20 77 6F 75 6C 64 20 6C 69 6B 65 20 74
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
40
08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00
00 28 B2 6C 40 00 80 06 92 1C 59 06 06 03 59 07
FE 36 09 53 00 6E 1A 5D 8A 76 50 DA 4E A5 50 10
22 38 89 41 00 00 00 00 00 00 00 00 BA 3C 6B D6
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-**
```

mfaptpkt2.txt

05

42

08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00
00 30 B0 6C 40 00 80 06 94 14 59 06 06 03 59 07
FE 36 09 53 00 6E 1A 5D 8A 6E 50 DA 49 60 50 18
1D 4B BF 97 00 00 52 45 54 52 20 32 0D 0A FD 6E
9D F5

4B

00 A0 24 75 C7 78 08 00 20 13 10 D2 08 00 45 00
00 39 16 03 00 00 3C 06 B2 75 59 07 FE 36 59 06
06 03 00 6E 09 53 50 DA 49 60 1A 5D 8A 76 50 18
10 00 5D 6C 00 00 2B 4F 4B 20 31 33 35 31 20 6F
63 74 65 74 73 0D 0A CA E0 6A B1

40

08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00
00 28 B1 6C 40 00 80 06 93 1C 59 06 06 03 59 07
FE 36 09 53 00 6E 1A 5D 8A 76 50 DA 49 71 50 10
1D 3A 93 73 00 00 00 00 00 00 00 00 02 03 21 C2

0460

00 A0 24 75 C7 78 08 00 20 13 10 D2 08 00 45 00
05 5C 16 05 00 00 3C 06 AD 50 59 07 FE 36 59 06
06 03 00 6E 09 53 50 DA 49 71 1A 5D 8A 76 50 18
10 00 53 11 00 00 52 65 74 75 72 6E 2D 50 61 74
68 3A 20 3C 6A 6D 65 74 7A 67 65 72 40 74 65 63
65 6C 69 74 65 2E 63 6F 6D 3E 0D 0A 52 65 63 65
69 76 65 64 3A 20 66 72 6F 6D 20 6E 61 74 61 64
6D 2E 74 65 63 65 6C 69 74 65 2E 63 6F 6D 20 62
79 20 73 75 70 65 72 2E 74 65 63 65 6C 69 74 65
2E 63 6F 6D 20 28 34 2E 31 2F 53 4D 49 2D 34 2E
31 29 0D 0A 09 69 64 20 41 41 32 38 34 30 38 3B
20 54 68 75 2C 20 31 30 20 53 65 70 20 39 38 20
31 37 3A 33 37 3A 33 37 20 50 44 54 0D 0A 52 65
63 65 69 76 65 64 3A 20 66 72 6F 6D 20 73 6D 74
70 6C 69 6E 6B 2E 74 65 63 65 6C 69 74 65 2E 63
6F 6D 20 28 73 6D 74 70 6C 69 6E 6B 20 5B 38 39
2E 37 2E 37 2E 31 30 30 5D 29 0D 0A 09 62 79 20
6E 61 74 61 64 6D 2E 74 65 63 65 6C 69 74 65 2E
63 6F 6D 20 28 38 2E 38 2E 37 2F 38 2E 38 2E 37
29 20 77 69 74 68 20 53 4D 54 50 20 69 64 20 52
41 41 31 37 32 34 35 3B 0D 0A 09 54 68 75 2C 20
31 30 20 53 65 70 20 31 39 39 38 20 31 37 3A 33
39 3A 30 34 20 2D 30 37 30 30 0D 0A 52 65 63 65
69 76 65 64 3A 20 66 72 6F 6D 20 63 63 3A 4D 61
69 6C 20 62 79 20 73 6D 74 70 6C 69 6E 6B 2E 74
65 63 65 6C 69 74 65 2E 63 6F 6D 0D 0A 09 69 64
20 41 41 39 30 35 34 37 34 38 32 39 20 54 68 75
2C 20 31 30 20 53 65 70 20 39 38 20 31 37 3A 34
37 3A 30 39 20 50 44 54 0D 0A 44 61 74 65 3A 20
54 68 75 2C 20 31 30 20 53 65 70 20 39 38 20 31
37 3A 34 37 3A 30 39 20 50 44 54 0D 0A 46 72 6F
6D 3A 20 4A 6F 68 6E 20 4D 65 74 7A 67 65 72 20
3C 6A 6D 65 74 7A 67 65 72 40 74 65 63 65 6C 69
74 65 2E 63 6F 6D 3E 0D 0A 45 6E 63 6F 64 69 6E
67 3A 20 33 32 34 20 54 65 78 74 0D 0A 4D 65 73
73 61 67 65 2D 49 64 3A 20 3C 39 38 30 38 31 30
39 30 35 34 2E 41 41 39 30 35 34 37 34 38 32 39
40 73 6D 74 70 6C 69 6E 6B 2E 74 65 63 65 6C 69
74 65 2E 63 6F 6D 3E 0D 0A 54 6F 3A 20 62 6C 65
61 76 79 40 74 65 63 65 6C 69 74 65 2E 63 6F 6D
2C 20 61 63 68 61 64 64 40 74 65 63 65 6C 69
74 65 2E 63 6F 6D 2C 20 64 61 76 65 63 40 74 65
63 65 6C 69 74 65 2E 63 6F 6D 2C 0D 0A 20 20 20
20 20 20 20 20 44 61 76 69 64 20 4C 75 6F 20 3C

mfaptpkt2.txt

```

64 6C 75 6F 40 74 65 63 65 6C 69 74 65 2E 63 6F
6D 3E 2C 20 6C 6F 77 64 65 72 40 74 65 63 65 6C
69 74 65 2E 63 6F 6D 2C 0D 0A 20 20 20 20 20 20
20 20 65 77 68 65 65 6C 65 72 40 74 65 63 65 6C
69 74 65 2E 63 6F 6D 2C 20 66 6E 6F 6F 6E 40 74
65 63 65 6C 69 74 65 2E 63 6F 6D 2C 20 66 72 65
64 6D 40 74 65 63 65 6C 69 74 65 2E 63 6F 6D 2C
0D 0A 20 20 20 20 20 20 20 20 20 6A 6D 61 69 78 6E
65 72 40 74 65 63 65 6C 69 74 65 2E 63 6F 6D 2C
20 6A 6F 74 69 73 40 74 65 63 65 6C 69 74 65 2E
63 6F 6D 2C 0D 0A 20 20 20 20 20 20 20 20 4B 69
6D 20 44 61 76 69 73 20 3C 6B 64 61 76 69 73 40
74 65 63 65 6C 69 74 65 2E 63 6F 6D 3E 2C 20 72
61 6D 40 74 65 63 65 6C 69 74 65 2E 63 6F 6D 2C
0D 0A 20 20 20 20 20 20 20 20 52 6F 62 20 52 69
74 7A 20 3C 72 72 69 74 7A 40 74 65 63 65 6C 69
74 65 2E 63 6F 6D 3E 2C 20 72 73 64 69 65 74 7A
40 74 65 63 65 6C 69 74 65 2E 63 6F 6D 2C 20 73
6B 69 70 40 74 65 63 65 6C 69 74 65 2E 63 6F 6D
0D 0A 53 75 62 6A 65 63 74 3A 20 4E 65 78 74 20
47 65 6E 65 72 61 74 69 6F 6E 20 50 72 6F 64 75
63 74 20 44 69 73 63 75 73 73 69 6F 6E 0D 0A 0D
0A 0D 0A 53 75 62 6A 65 63 74 3A 20 4E 65 78 74
20 47 65 6E 65 72 61 74 69 6F 6E 20 50 72 6F 64
75 63 74 20 44 69 73 63 75 73 73 69 6F 6E 0D 0A
0D 0A 49 20 77 6F 75 6C 64 20 6C 69 6B 65 20 74
40
08 00 20 13 10 D2 00 A0 24 75 C7 78 08 00 45 00
00 28 B2 6C 40 00 80 06 92 1C 59 06 06 03 59 07
FE 36 09 53 00 6E 1A 5D 8A 76 50 DA 4E A5 50 10
22 38 89 41 00 00 00 00 00 00 00 00 BA 3C 6B D6

```

- **Exhibit B6:** The contents of files mfaptkey.txt and mfaptkey2.txt that are files that contain the keys that were generated from the extracted data (Exhibit B4) and used for looking up the flow-entry database per element (c) of method claims 11 and 54, which are operations carried out by the lookup engine of element (e) of claim 29.

[illegible]

mfaptkey2.txt

05

00 00 00 00 08 00 20 13 10 D2 00 A0 24 75 C7 78
00 08 59 07 FE 36 00 00 00 00 00 00 00 00 00
00 00 59 06 06 03 00 00 00 00 00 00 00 00 00
00 00 00 28 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 47 00 6E 00 00 09 53 00 00

00 00 00 00 00 A0 24 75 C7 78 08 00 20 13 10 D2
00 08 59 06 06 03 00 00 00 00 00 00 00 00 00
00 00 59 07 FE 36 00 00 00 00 00 00 00 00 00 00
00 00 00 28 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 47 09 53 00 00 00 6E 00 00

00 00 00 00 08 00 20 13 10 D2 00 A0 24 75 C7 78
00 08 59 07 FE 36 00 00 00 00 00 00 00 00 00 00
00 00 59 06 06 03 00 00 00 00 00 00 00 00 00
00 00 00 28 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 47 00 6E 00 00 09 53 00 00

00 00 00 00 00 A0 24 75 C7 78 08 00 20 13 10 D2
00 08 59 06 06 03 00 00 00 00 00 00 00 00 00
00 00 59 07 FE 36 00 00 00 00 00 00 00 00 00 00
00 00 00 28 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 47 09 53 00 00 00 6E 00 00

00 00 00 00 08 00 20 13 10 D2 00 A0 24 75 C7 78
00 08 59 07 FE 36 00 00 00 00 00 00 00 00 00 00
00 00 59 06 06 03 00 00 00 00 00 00 00 00 00
00 00 00 28 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 47 00 6E 00 00 09 53 00 00

- **Exhibit B7:** The contents of file MFATEST.TXT that includes the decoded packets that were generated by operation of the method that includes the elements of each of method claims 11 and 54, by an apparatus that includes the elements of claim 29.

1 TO: 0800201310D2 [REDACTED] 22:53:50<0.000> [REDACTED]
FROM: 00A02475C778

[illegible][illegible][illegible]

```
Pkt: 4, Len: 1120/1390
Ethernet: (Sun 1310d2 -> 00a02475c778) type: IP(0x800)
Internet: 89.7.254.54 -> 89.6.6.3 hl: 5 ver: 4
tos: 0 len: 1372 id: 0x1605 fragoff: 0 flags: 00 ttl: 60
prot: TCP(6) xsum: 0xad50
TCP: POP-3(110) -> 2387 seq: 50da4971
ack: 1a5d8a76 win: 4096 hl: 5 xsum: 0x5311 urg: 0
flags: <ACK><PUSH>
```

data (60/1332):

--*-*-*

5 TO: 0800201310D2 22:53:51<0.198>
FROM: 00A02475C778

Pkt: 5, Len: 64/64
Ethernet: (00a02475c778 -> Sun 1310d2) type: IP(0x800)
Internet: 89.6.6.3 -> 89.7.254.54 hl: 5 ver: 4
tos: 0 len: 40 id: 0xb26c fragoff: 0 flags: 0x2 ttl: 128
prot: TCP(6) xsum: 0x921c
TCP: 2387 -> POP-3(110) seq: 1a5d8a76
ack: 50da4ea5 win: 8760 hl: 5 xsum: 0x8941 urg: 0
flags: <ACK>

--*-*-*

6 TO: 0800201310D2 22:53:53<2.556>
FROM: 00A02475C778

Pkt: 6, Len: 66/66
Ethernet: (00a02475c778 -> Sun 1310d2) type: IP(0x800)
Internet: 89.6.6.3 -> 89.7.254.54 hl: 5 ver: 4
tos: 0 len: 48 id: 0xb36c fragoff: 0 flags: 0x2 ttl: 128
prot: TCP(6) xsum: 0x9114
TCP: 2387 -> POP-3(110) seq: 1a5d8a76
ack: 50da4ea5 win: 8760 hl: 5 xsum: 0xcb6a urg: 0
flags: <ACK><PUSH>
data (8/8):

--*-*-*

7 TO: 00A02475C778 22:53:53<0.001>
FROM: 0800201310D2

Pkt: 7, Len: 91/91
Ethernet: (Sun 1310d2 -> 00a02475c778) type: IP(0x800)
Internet: 89.7.254.54 -> 89.6.6.3 hl: 5 ver: 4
tos: 0 len: 73 id: 0x1609 fragoff: 0 flags: 00 ttl: 60
prot: TCP(6) xsum: 0xb25f
TCP: POP-3(110) -> 2387 seq: 50da4ea5
ack: 1a5d8a7e win: 4096 hl: 5 xsum: 0x3f4c urg: 0
flags: <ACK><PUSH>
data (33/33):

--*-*-*

8 TO: 0800201310D2 22:53:53<0.002>
FROM: 00A02475C778

Pkt: 8, Len: 64/64
Ethernet: (00a02475c778 -> Sun 1310d2) type: IP(0x800)
Internet: 89.6.6.3 -> 89.7.254.54 hl: 5 ver: 4
tos: 0 len: 46 id: 0xb46c fragoff: 0 flags: 0x2 ttl: 128
prot: TCP(6) xsum: 0x9016
TCP: 2387 -> POP-3(110) seq: 1a5d8a7e
ack: 50da4ec6 win: 8727 hl: 5 xsum: 0xe177 urg: 0

[illegible]

22:53:53<0.029>

* _ * _ * _ * _ * _ * _ * _ * _ * _ * _ * _ * _ * _ * _ * _ *

22:53:53<0.003>

[illegible]

22:53:53<0.000>

[illegible]

██████████:53:53<0.031> ██████████

```
Pkt: 12, Len: 64/64
Ethernet: (00a02475c778 -> Sun 1310d2) type: IP(0x800)
Internet: 89.6.6.3 -> 89.7.254.54 hl: 5 ver: 4
tos: 0 len: 40 id: 0xb66c fragoff: 0 flags: 0x2 ttl: 128
prot: TCP(6) xsum: 0x8e1c
TCP: 2387 -> POP-3(110) seq: 1a5d8a84
```


[illegible]

22:53:53<0.001>

[illegible]

22:53:58<4.755>

_

22:53:58<0.001>

[illegible]

22:53:58<0.000>

```
Pkt: 16, Len: 64/64
Ethernet: (00a076a010f2 -> 006008c0d710) type: IP(0x800)
Internet: 89.89.24.6 -> 89.75.23.24 hl: 5 ver: 4
tos: 0 len: 40 id: 0x60fe fragoff: 0 flags: 0x2 ttl: 128
prot: TCP(6) xsum: 0xb80f
```

- **Exhibit B8:** Protocol Definition Language (PDL) Reference Guide (the document MFS-PDL-Reference.pdf) that provides a reference to the protocol definition language used in cpl files.

MeterFlow™ Traffic Classification System

Protocol Definition Language (PDL) Reference Guide

Version A0.02



DRAFT - 02



Technically Elite

monitoring your enterprise

NOTICE

This document contains confidential information proprietary to Technically Elite, Inc.

No part of its content may be used, copied, disclosed or conveyed to any party in any manner without prior written permission of Technically Elite, Inc.

Restricted Rights Legend

The programs and information contained herein are licensed only pursuant to a license agreement that contains use, reverse engineering, disclosure and other restrictions; accordingly, it is "Unpublished - all rights reserved under the applicable copyright laws."

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Licensed Programs clause at DFARS 52.227-7013.

Copyright © [REDACTED] by Technically Elite, Inc.

All Rights Reserved.

Printed in the United States of America.

Government Use

The Licensed Programs and their documentation were developed at private expense and no part of this is in the public domain.

The Licensed Programs are "Restricted Computer Software" as that term is defined in Clause 52.227-19 of the Federal Acquisition Regulations (FAR) and are "Commercial Computer Software" as that term is defined in Subpart 227.401 of the Department of Defense Federal Acquisition Regulation Supplement (DFARS).

- (i) If the licensed Programs are supplied to the Department of Defense (DoD), the Licensed Programs are classified as "Commercial computer Software" and the Government is acquiring only "restricted rights" in the Licensed Programs and their documentation as that term is defined in Clause 52.227-7013(c)(1) of the DFARS, and
- (ii) If the Licensed Programs are supplied to any unit or agency of the United States Government other than DoD, the Government's rights in the Licensed Programs and their documentation will be defined in Clause 52.227-19(c)(2) of the FAR.

All Technically Elite product names are trademarks or registered trademarks of Technically Elite, Inc. Other product names used in the manual are trademarks or registered trademarks of their respective holders.

Document revision A.02-004, format revision 1.00-000.

Technically Elite, Inc.

6330 San Ignacio Ave.

San Jose, CA 95119-1209, USA

Phone: +1.408.574.2300

Fax: +1.408.629.8300

E-mail: mworks@tecelite.com

URL: <http://www.tecelite.com>

Contents

1. INTRODUCTION	5
1.1 SUMMARY	5
1.2 DOCUMENT CONVENTIONS	5
2. LANGUAGE STRUCTURE	7
3. PROGRAM STRUCTURE	7
3.1 FIELD DEFINITIONS	7
3.1.1 SYNTAX Type [{ Enums }]	7
3.1.2 DISPLAY-HINT "FormatString"	8
3.1.3 LENGTH "Expression"	8
3.1.4 FLAGS FieldFlags	8
3.1.5 ENCAP FieldName [, FieldName2]	9
3.1.6 LOOKUP LookupType [Filename]	9
3.1.7 ENCODING EncodingType	9
3.1.8 DEFAULT "value"	9
3.1.9 DESCRIPTION "Description"	9
3.2 GROUP DEFINITIONS	10
3.2.1 LENGTH "Expression"	10
3.2.2 OPTIONAL "Condition"	10
3.2.3 SUMMARIZE "Condition" : "FormatString" ["Condition" : "FormatString"...]	10
3.2.4 DESCRIPTION "Description"	11
3.2.5 ::= { Name=FieldOrGroup [, Name=FieldOrGroup...] }	11
3.3 PROTOCOL DEFINITIONS	12
3.3.1 SUMMARIZE "Condition" : "FormatString" ["Condition" : "FormatString"...]	12
3.3.2 DESCRIPTION "Description"	12
3.3.3 REFERENCE "Reference"	12
3.3.4 ::= { Name=FieldOrGroup [, Name=FieldOrGroup...] }	12
3.4 FLOW DEFINITIONS	13
3.4.1 HEADER { Option [, Option...] }	13
3.4.2 DLC-LAYER { Option [, Option...] }	13
3.4.3 NET-LAYER { Option [, Option...] }	13
3.4.4 CONNECTION { Option [, Option...] }	14
3.4.5 PAYLOAD { Option [, Option...] }	14
3.4.6 CHILDREN { Option [, Option...] }	14
3.4.7 STATE-BASED	14
3.4.8 STATES "Definitions"	15
3.5 CONDITIONS	16
3.6 STATE DEFINITIONS	17

3.6.1	<i>CHECKCONNECT, operand</i>	17
3.6.2	<i>GOTO state</i>	17
3.6.3	<i>NEXT state</i>	17
3.6.4	<i>DEFAULT operand</i>	17
3.6.5	<i>CHILD protocol</i>	17
3.6.6	<i>WAIT numPackets, operand1, operand2</i>	17
3.6.7	<i>MATCH 'string' weight offset LF-offset range LF-range, operand</i>	17
3.6.8	<i>CONSTANT number offset range, operand</i>	17
3.6.9	<i>EXTRACTIP offset destination, operand</i>	17
3.6.10	<i>EXTRACTPORT offset destination, operand</i>	17
3.6.11	<i>CREATEREDIRECTEDFLOW, operand</i>	18
4.	EXAMPLE PDL RULES	19
4.1	ETHERNET	19
4.2	IP VERSION 4	20
4.3	TCP.....	22
4.4	HTTP (WITH STATE).....	24
5.	SUPPLEMENTAL PRODUCTS	27

1. Introduction

The *MeterFlow* Protocol Definition Language (PDL) is a special purpose language used to describe network protocols and all the fields within the protocol headers.

Within this document, protocol descriptions (PDL files) are referred to as *PDL* or *rules* when there is no risk of confusion with other types of descriptions.

PDL uses both form and organization similar to the data structure definition part of the C programming language and the PERL scripting language. Since PDL was derived from a language used to decode network packet content, the authors have mixed the language format with the requirements of packet decoding. This results in an expressive language that is very familiar and comfortable for describing packet content and the details required representing a flow.

1.1 Summary

MeterFlow PDL is a non-procedural Forth Generation language (4GL). This means it describes *what* needs to be done without describing *how* to do it. The details of *how* are hidden in the compiler and the Compiled Protocol Layout (CPL) optimization utility.

In addition, it is used to describe network flows by defining which fields are the address fields, which are the protocol type fields, etc.

Once a PDL file is written, it is compiled using the Netscope compiler (**nsc**), which produces the *MeterFlow* database (MeterFlow.db) and the Netscope database (Netscope.db). The MeterFlow database contains the flow definitions and the Netscope database contains the protocol header definitions.

These databases are used by programs like: **mfkeys**, which produces flow keys; **mfcp**, which produces flow definitions in CPL format; **mfpkts** which produces sample packets of all known protocols; and **netscope**, which decodes Sniffer™ and tcpdump files.

Due to its size, electronic media copies of the documentation are not provided but can be made available if necessary.

1.2 Document Conventions

The following conventions will be used throughout this document:

Small `courier` typeface indicates C code examples or function names. Functions are written with parentheses after them [**function()**], variables are written just as their names [**variables**], and structure names are written prefixed with “**struct**” [**struct packet**].

Italics indicate a filename (for instance, *mworks/base/h/base.h*). Filenames will usually be written relative to the root directory of the distribution.

Constants are expressed in decimal, unless written “0x . . .”, the C language notation for hexadecimal numbers.

2. Language Structure

TBD

3. Program Structure

A *MeterFlow* PDL decodes and flow set is a non-empty sequence of statements.

There are four basic types of statements or definitions available in *MeterFlow* PDL:

FIELD,
GROUP,
PROTOCOL and
FLOW.

3.1 FIELD Definitions

The **FIELD** definition is used to define a specific string of bits or bytes in the packet. The **FIELD** definition has the following format:

```
Name      FIELD
          SYNTAX Type [ { Enums } ]
          DISPLAY-HINT "FormatString"
          LENGTH "Expression"
          FLAGS FieldFlags
          ENCAP FieldName [ , FieldName2 ]
          LOOKUP LookupType [ Filename ]
          ENCODING EncodingType
          DEFAULT "value"
          DESCRIPTION "Description"
```

Where only the **FIELD** and **SYNTAX** lines are required. All the other lines are attribute lines, which define special characteristics about the **FIELD**. Attribute lines are optional and may appear in any order. Each of the attribute lines are described in detail below:

3.1.1 SYNTAX Type [{ Enums }]

This attribute defines the type and, if the type is an INT, BYTESTRING, BITSTRING, or SNMPSEQUENCE type, the enumerated values for the **FIELD**. The currently defined types are:

INT(<i>numBits</i>)	Integer that is <i>numBits</i> bits long.
UNSIGNED INT(<i>numBits</i>)	Unsigned integer that is <i>numBits</i> bits long.

BYTESTRING(<i>numBytes</i>)	String that is <i>numBytes</i> bytes long.
BYTESTRING(<i>R1</i> .. <i>R2</i>)	String that ranges in size from <i>R1</i> to <i>R2</i> bytes.
BITSTRING(<i>numBits</i>)	String that is <i>numBits</i> bits long.
LSTRING(<i>lenBytes</i>)	String with <i>lenBytes</i> header.
NSTRING	Null terminated string.
DNSSTRING	DNS encoded string.
SNMPOID	SNMP Object Identifier.
SNMPSEQUENCE	SNMP Sequence.
SNMPTIMETICKS	SNMP TimeTicks.
COMBO <i>field1 field2</i>	Combination pseudo field.

3.1.2 DISPLAY-HINT "FormatString"

This attribute is for specifying how the value of the FIELD is displayed. The currently supported formats are:

Numx	Print as a num byte hexadecimal number.
Numd	Print as a num byte decimal number.
Numo	Print as a num byte octal number.
Numb	Print as a num byte binary number.
Numa	Print num bytes in ASCII format.
Text	Print as ASCII text.
HexDump	Print in hexdump format.

3.1.3 LENGTH "Expression"

This attribute defines an expression for determining the FIELD's length. Expressions are arithmetic and can refer to the value of other FIELD's in the packet by adding a \$ to the referenced field's name. For example, "(\$tcpHeaderLen *4) - 20" is a valid expression if tcpHeaderLen is another field defined for the current packet.

3.1.4 FLAGS FieldFlags

The attribute defines some special flags for a FIELD. The currently supported FieldFlags are:

SAMELAYER	Display field on the same layer as the previous field.
NOLABEL	Don't display the field name with the value.

NOSHOW	Decode the field but don't display it.
SWAPPED	The integer value is swapped.

3.1.5 ENCAP FieldName [, FieldName2]

This attribute defines how one packet is encapsulated inside another. Which packet is determined by the value of the FieldName field. If no packet is found using FieldName then FieldName2 is tried.

3.1.6 LOOKUP LookupType [Filename]

This attribute defines how to lookup the name for a particular FIELD value. The currently supported LookupTypes are:

SERVICE	Use getservbyport().
HOSTNAME	Use gethostbyaddr().
MACADDRESS	Use \$METERFLOW/conf/mac2ip.cf.
FILE <i>file</i>	Use <i>file</i> to lookup value.

3.1.7 ENCODING EncodingType

This attribute defines how a FIELD is encoded. Currently, the only supported EncodingType is BER (for Basic Encoding Rules defined by ASN.1).

3.1.8 DEFAULT "value"

This attribute defines the default value to be used for this field when generating sample packets of this protocol.

3.1.9 DESCRIPTION "Description"

This attribute defines the description of the FIELD. It is used for informational purposes only.

3.2 GROUP Definitions

The GROUP definition is used to tie several related FIELDS together. The GROUP definition has the following format:

```

Name      GROUP
          LENGTH "Expression"
          OPTIONAL "Condition"
          SUMMARIZE "Condition" : "FormatString" [
            "Condition" : "FormatString"... ]
          DESCRIPTION "Description"
          ::= { Name=FieldOrGroup [ ,
            Name=FieldOrGroup... ] }

```

Where only the GROUP and ::= lines are required. All the other lines are attribute lines, which define special characteristics for the GROUP. Attribute lines are optional and may appear in any order. Each attribute line is described in detail below:

3.2.1 LENGTH "Expression"

This attribute defines an expression for determining the GROUP's length. Expressions are arithmetic and can refer to the value of other FIELD's in the packet by adding a \$ to the referenced field's name. For example, "(\$tcpHeaderLen *4) – 20" is a valid expression if tcpHeaderLen is another field defined for the current packet.

3.2.2 OPTIONAL "Condition"

This attribute defines a condition for determining whether a GROUP is present or not. Valid conditions are defined in the Conditions section below.

3.2.3 SUMMARIZE "Condition" : "FormatString" ["Condition" : "FormatString"...]

This attribute defines how a GROUP will be displayed in Detail mode. A different format (FormatString) can be specified for each condition (Condition). Valid conditions are defined in the Conditions section below. Any FIELD's value can be referenced within the FormatString by proceeding the FIELD's name with a \$. In addition to FIELD names there are several other special \$ keywords:

\$LAYER	Displays the current protocol layer.
\$GROUP	Displays the entire GROUP as a table.
\$LABEL	Displays the GROUP label.
\$ <i>field</i>	Displays the <i>field</i> value (use enumerated name if available).
\$. <i>field</i>	Displays the <i>field</i> value (in raw format).

3.2.4 DESCRIPTION "Description"

This attribute defines the description of the GROUP. It is used for informational purposes only.

3.2.5 ::= { Name=FieldOrGroup [, Name=FieldOrGroup...] }

This defines the order of the fields and subgroups within the GROUP.

3.3 PROTOCOL Definitions

The PROTOCOL definition is used to define the order of the FIELDS and GROUPs within the protocol header. The PROTOCOL definition has the following format:

```

Name      PROTOCOL
SUMMARIZE "Condition" : "FormatString" [
"Condition" : "FormatString"... ]
DESCRIPTION "Description"
REFERENCE "Reference"
::= { Name=FieldOrGroup [ ,
Name=FieldOrGroup... ] }

```

Where only the PROTOCOL and ::= lines are required. All the other lines are attribute lines, which define special characteristics for the PROTOCOL. Attribute lines are optional and may appear in any order. Each attribute line is described in detail below:

3.3.1 SUMMARIZE "Condition" : "FormatString" ["Condition" : "FormatString"...]

This attribute defines how a PROTOCOL will be displayed in Summary mode. A different format (FormatString) can be specified for each condition (Condition). Valid conditions are defined in the Conditions section below. Any FIELD's value can be referenced within the FormatString by proceeding the FIELD's name with a \$. In addition to FIELD names there are several other special \$ keywords:

\$LAYER	Displays the current protocol layer.
\$VARBIND	Displays the entire SNMP VarBind list.
\$ <i>field</i>	Displays the <i>field</i> value (use enumerated name if available).
\$. <i>field</i>	Displays the <i>field</i> value (in raw format).
\$# <i>field</i>	Counts all occurrences of <i>field</i> .
\$* <i>field</i>	Lists all occurrences of <i>field</i> .

3.3.2 DESCRIPTION "Description"

This attribute defines the description of the PROTOCOL. It is used for informational purposes only.

3.3.3 REFERENCE "Reference"

This attribute defines the reference material used to determine the protocol format. It is used for informational purposes only.

3.3.4 ::= { Name=FieldOrGroup [, Name=FieldOrGroup...] }

This defines the order of the FIELDS and GROUPs within the PROTOCOL.

3.4 FLOW Definitions

The FLOW definition is used to define a network flow by describing where the address, protocol type, and port numbers are in a packet. The FLOW definition has the following format:

```

Name      FLOW
          HEADER { Option [, Option...] }
          DLC-LAYER { Option [, Option...] }
          NET-LAYER { Option [, Option...] }
          CONNECTION { Option [, Option...] }
          PAYLOAD { Option [, Option...] }
          CHILDREN { Option [, Option...] }
          STATE-BASED
          STATES "Definitions"

```

Where only the FLOW line is required. All the other lines are attribute lines, which define special characteristics for the FLOW. Attribute lines are optional and may appear in any order. However, at least one attribute line must be present. Each attribute line is described in detail below:

3.4.1 HEADER { Option [, Option...] }

This attribute is used to describe the length of the protocol header. The currently supported Options are:

LENGTH= <i>number</i>	Header is a fixed length of size <i>number</i> .
LENGTH= <i>field</i>	Header is variable length determined by value of <i>field</i> .
IN-WORDS	The units of the header length are in 32-bit words rather than bytes.

3.4.2 DLC-LAYER { Option [, Option...] }

If the protocol is a data link layer protocol, this attribute describes it. The currently supported Options are:

DESTINATION= <i>field</i>	Indicates which <i>field</i> is the DLC destination address.
SOURCE= <i>field</i>	Indicates which <i>field</i> is the DLC source address.
PROTOCOL	Indicates this is a data link layer protocol.
TUNNELING	Indicates this is a tunneling protocol.

3.4.3 NET-LAYER { Option [, Option...] }

If the protocol is a network layer protocol, then this attribute describes it. The currently supported Options are:

DESTINATION= <i>field</i>	Indicates which <i>field</i> is the network destination address.
---------------------------	--

SOURCE= <i>field</i>	Indicates which <i>field</i> is the network source address.
TUNNELING	Indicates this is a tunneling protocol.
FRAGMENTATION= <i>type</i>	Indicates this protocol supports fragmentation. There are currently two fragmentation types: IPV4 and IPV6.

3.4.4 CONNECTION { Option [, Option...] }

If the protocol is a connection-oriented protocol, then this attribute describes how connections are established and torn down. The currently supported Options are:

IDENTIFIER= <i>field</i>	Indicates the connection identifier <i>field</i> .
CONNECT-START=" <i>flag</i> "	Indicates when a connection is being initiated.
CONNECT-COMPLETE=" <i>flag</i> "	Indicates when a connection has been established.
DISCONNECT-START=" <i>flag</i> "	Indicates when a connection is being torn down.
DISCONNECT-COMPLETE=" <i>flag</i> "	Indicates when a connection has been torn down.
INHERITED	Indicates this is a connection-oriented protocol but the parent protocol is where the connection is established.

3.4.5 PAYLOAD { Option [, Option...] }

This attribute describes how much of the payload from a packet of this type should be stored for later use during analysis. The currently supported Options are:

INCLUDE-HEADER	Indicates that the protocol header should be included.
LENGTH= <i>number</i>	Indicates how many bytes of the payload should be stored.
DATA= <i>field</i>	Indicates which <i>field</i> contains the payload.

3.4.6 CHILDREN { Option [, Option...] }

This attribute describes how children protocols are determined. The currently supported Options are:

DESTINATION= <i>field</i>	Indicates which <i>field</i> is the destination port.
SOURCE= <i>field</i>	Indicates which <i>field</i> is the source port.
LLCCHECK= <i>flow</i>	Indicates that if the DESTINATION field is less than 0x05DC then use <i>flow</i> instead of the current flow definition.

3.4.7 STATE-BASED

This attribute indicates that the flow is a state-based flow.

3.4.8 STATES “Definitions”

This attribute describes how children flows of this protocol are determined using states. See the State Definitions section below for how these states are defined.

3.5 CONDITIONS

Conditions are used with the OPTIONAL and SUMMARIZE attributes and may consist of the following:

Value1 == Value2	Value1 equals Value2. Works with string values.
Value1 != Value2	Value1 does not equal Value2. Works with string values.
Value1 <= Value2	Value1 is less than or equal to Value2.
Value1 >= Value2	Value1 is greater than or equal to Value2.
Value1 < Value2	Value1 is less than Value2.
Value1 > Value2	Value1 is greater than Value2.
Field m/regex/	Field matches the regular expression regex.

Where *Value1* and *Value2* can be either FIELD references (field names preceded by a \$) or constant values. Note that compound conditional statements (using AND and OR) are not currently supported.

3.6 STATE DEFINITIONS

Many applications running over data networks utilize complex methods of classifying traffic through the use of multiple states. State definitions are used for managing and maintaining learned states from traffic derived from the network.

The basic format of a state definition is:

StateName: Operand Parameters [Operand Parameters...]

The various states of a particular flow are described using the following operands:

3.6.1 CHECKCONNECT, *operand*

Checks for connection. Once connected executes *operand*.

3.6.2 GOTO *state*

Goes to *state*, using the current packet.

3.6.3 NEXT *state*

Goes to *state*, using the next packet.

3.6.4 DEFAULT *operand*

Executes *operand* when all other operands fail.

3.6.5 CHILD *protocol*

Jump to child *protocol* and perform state-based processing (if any) in the child.

3.6.6 WAIT *numPackets*, *operand1*, *operand2*

Waits the specified number of packets. Executes *operand1* when the specified number of packets have been received. Executes *operand2* when a packet is received but it is less than the number of specified packets.

3.6.7 MATCH '*string*' *weight offset LF-offset range LF-range*, *operand*

Searches for a *string* in the packet, executes *operand* if found.

3.6.8 CONSTANT *number offset range*, *operand*

Checks for a constant in a packet, executes *operand* if found.

3.6.9 EXTRACTIP *offset destination*, *operand*

Extracts an IP address from the packet and then executes *operand*.

3.6.10 EXTRACTPORT *offset destination*, *operand*

Extracts a port number from the packet and then executes *operand*.

3.6.11 CREATEREDIRECTEDFLOW, *operand*

Creates a redirected flow and then executes *operand*.

4. Example PDL Rules

The following section contains several examples of PDL Rule files.

4.1 Ethernet

The following is an example of the PDL for Ethernet:

```
MacAddress FIELD
    SYNTAX          BYTESTRING(6)
    DISPLAY-HINT    "1x:"
    LOOKUP          MACADDRESS
    DESCRIPTION
        "MAC layer physical address"

etherType FIELD
    SYNTAX          INT(16)
    DISPLAY-HINT    "1x:"
    LOOKUP          FILE "EtherType.cf"
    DESCRIPTION
        "Ethernet type field"

etherData FIELD
    SYNTAX          BYTESTRING(46..1500)
    ENCAP          etherType
    DISPLAY-HINT    "HexDump"
    DESCRIPTION
        "Ethernet data"

ethernet PROTOCOL
    DESCRIPTION
        "Protocol format for an Ethernet frame"
    REFERENCE "RFC 894"
    ::= { MacDest=macAddress, MacSrc=macAddress,
        EtherType=etherType,
        Data=etherData }

ethernet FLOW
    HEADER { LENGTH=14 }
    DLC-LAYER {
        SOURCE=MacSrc,
        DESTINATION=MacDest,
        TUNNELING,
        PROTOCOL
    }
    CHILDREN { DESTINATION=EtherType, LLC-CHECK=llc }
```

4.2 IP Version 4

Here is an example of the PDL for the IP protocol:

```

ipAddress  FIELD
           SYNTAX          BYTESTRING(4)
           DISPLAY-HINT    "1d."
           LOOKUP           HOSTNAME
           DESCRIPTION
             "IP address"

ipVersion  FIELD
           SYNTAX          INT(4)
           DEFAULT         "4"

ipHeaderLength  FIELD
                SYNTAX          INT(4)

ipTypeOfService  FIELD
                 SYNTAX          BITSTRING(8) { minCost(1),
                                                maxReliability(2), maxThruput(3),
                                                minDelay(4) }

ipLength        FIELD
                 SYNTAX          UNSIGNED INT(16)

ipFlags          FIELD
                 SYNTAX          BITSTRING(3) { moreFrag(0),
dontFrag(1) }

IpFragmentOffset      FIELD
                       SYNTAX          INT(13)

ipProtocol  FIELD
            SYNTAX          INT(8)
            LOOKUP          FILE "IpProtocol.cf"

ipData      FIELD
            SYNTAX          BYTESTRING(0..1500)
            ENCAP           ipProtocol
            DISPLAY-HINT    "HexDump"

ip  PROTOCOL
    SUMMARIZE
    "$FragmentOffset != 0":
        "IPFragment ID=$Identification
Offset=$FragmentOffset"
    "Default" :
        "IP Protocol=$Protocol"
    DESCRIPTION
        "Protocol format for the Internet Protocol"
    REFERENCE "RFC 791"
::= { Version=ipVersion, HeaderLength=ipHeaderLength,
      TypeOfService=ipTypeOfService, Length=ipLength,

```

```

    Identification=UInt16, IpFlags=ipFlags,
    FragmentOffset=ipFragmentOffset, TimeToLive=Int8,
    Protocol=ipProtocol, Checksum=ByteStr2,
    IpSrc=ipAddress, IpDest=ipAddress, Options=ipOptions,
    Fragment=ipFragment, Data=ipData }

ip    FLOW
    HEADER { LENGTH=HeaderLength, IN-WORDS }
    NET-LAYER {
        SOURCE=IpSrc,
        DESTINATION=IpDest,
        FRAGMENTATION=IPV4,
        TUNNELING
    }
    CHILDREN { DESTINATION=Protocol }

ipFragData FIELD
    SYNTAX          BYTESTRING(1..1500)
    LENGTH          "ipLength - ipHeaderLength * 4"
    DISPLAY-HINT    "HexDump"

ipFragment GROUP
    OPTIONAL "$FragmentOffset != 0"
::= { Data=ipFragData }

ipOptionCode    FIELD
    SYNTAX          INT(8) { ipRR(0x07),
ipTimestamp(0x44),
                ipLSRR(0x83), ipSSRR(0x89) }
    DESCRIPTION
        "IP option code"

ipOptionLength  FIELD
    SYNTAX          UNSIGNED INT(8)
    DESCRIPTION
        "Length of IP option"

ipOptionData    FIELD
    SYNTAX          BYTESTRING(0..1500)
    ENCAP          ipOptionCode
    DISPLAY-HINT    "HexDump"

ipOptions GROUP
    LENGTH          "(ipHeaderLength * 4) - 20"
::= { Code=ipOptionCode, Length=ipOptionLength, Pointer=UInt8,
    Data=ipOptionData }

```


4.3 TCP

Here is an example of the PDL for the TCP protocol:

```

tcpPort FIELD
    SYNTAX      UNSIGNED INT(16)
    LOOKUP      FILE "TcpPort.cf"

tcpHeaderLen FIELD
    SYNTAX      INT(4)

tcpFlags FIELD
    SYNTAX      BITSTRING(12) { fin(0), syn(1), rst(2), psh(3),
                                ack(4), urg(5) }

tcpData FIELD
    SYNTAX      BYTESTRING(0..1564)
    LENGTH      "($ipLength-($ipHeaderLength*4))-
($tcpHeaderLen*4)"
    ENCAP       tcpPort
    DISPLAY-HINT "HexDump"

tcp  PROTOCOL
    SUMMARIZE
        "Default" :
            "TCP ACK=$Ack WIN=$WindowSize"
    DESCRIPTION
        "Protocol format for the Transmission Control
Protocol"
    REFERENCE  "RFC 793"
    ::= { SrcPort=tcpPort, DestPort=tcpPort, SequenceNum=UInt32,
Ack=UInt32, HeaderLength=tcpHeaderLen, TcpFlags=tcpFlags,
WindowSize=UInt16, Checksum=ByteStr2,
UrgentPointer=UInt16, Options=tcpOptions, Data=tcpData }

tcp  FLOW
    HEADER { LENGTH=HeaderLength, IN-WORDS }
    CONNECTION {
        IDENTIFIER=SequenceNum,
        CONNECT-START="TcpFlags:1",
        CONNEG-COMPLETE="TcpFlags:4",
        DISCONNECT-START="TcpFlags:0",
        DISCONNECT-COMPLETE="TcpFlags:4"
    }
    PAYLOAD { INCLUDE-HEADER }
    CHILDREN { DESTINATION=DestPort, SOURCE=SrcPort }

tcpOptionKind FIELD
    SYNTAX      UNSIGNED INT(8) { tcpOptEnd(0), tcpNop(1),
                                tcpMSS(2), tcpWscale(3),
                                tcpTimestamp(4) }
    DESCRIPTION
        "Type of TCP option"

```

```
tcpOptionData    FIELD
                  SYNTAX      BYTESTRING(0..1500)
                  ENCAP       tcpOptionKind
                  FLAGS       SAMELAYER
                  DISPLAY-HINT "HexDump"

tcpOptions GROUP
              LENGTH          "($tcpHeaderLen * 4) - 20"
::= { Option=tcpOptionKind, OptionLength=UInt8,
      OptionData=tcpOptionData }

tcpMSS        PROTOCOL
::= { MaxSegmentSize=UInt16 }
```

4.4 HTTP (with State)

Here is an example of the PDL for the HTTP protocol:

```

httpData FIELD
    SYNTAX  BYTESTRING(1..1500)
    LENGTH  "($ipLength - ($ipHeaderLength * 4)) - ($tcpHeaderLen
* 4)"
    DISPLAY-HINT "Text"
    FLAGS      NOLABEL

http      PROTOCOL
          SUMMARIZE
            "$httpData m/^(GET|^(HTTP|^(HEAD|^(POST/" :
              "HTTP $httpData"
            "$httpData m/^[Dd]ate|^[Ss]erver|^[Ll]ast-
[Mm]odified/" :
              "HTTP $httpData"
            "$httpData m/^[Cc]ontent-/" :
              "HTTP $httpData"
            "$httpData m/^(<HTML>/" :
              "HTTP [HTML document]"
            "$httpData m/^(GIF/" :
              "HTTP [GIF image]"
            "Default" :
              "HTTP [Data]"
          DESCRIPTION
            "Protocol format for HTTP."
      ::= { Data=httpData }

http FLOW
    HEADER { LENGTH=0 }
    CONNECTION { INHERITED }
    PAYLOAD { INCLUDE-HEADER, DATA=Data, LENGTH=256 }
    STATES
      "S0: CHECKCONNECT, GOTO S1
        DEFAULT NEXT S0

      S1: WAIT 2, GOTO S2, NEXT S1
        DEFAULT NEXT S0

      S2: MATCH
          '\n\r\n'          900 0 0 255 0, NEXT S3
          '\n\n'            900 0 0 255 0, NEXT S3
          'POST /tds?'       50 0 0 127 1, CHILD
sybaseWebsql
          '.hts HTTP/1.0'    50 4 0 127 1, CHILD
sybaseJdbc
          'jdbc:sybase:Tds'  50 4 0 127 1, CHILD
sybaseTds
          'PCN-The Poin'    500 4 1 255 0, CHILD
pointcast
          't: BW-C-'        100 4 1 255 0, CHILD backweb
          DEFAULT NEXT S3

```

```

S3: MATCH
    '\n\r\n'          50 0 0 0 0, NEXT S3
    '\n\n'            50 0 0 0 0, NEXT S3
    'Content-Type:'    800 0 0 255 0, CHILD mime
    'PCN-The Poin'     500 4 1 255 0, CHILD
pointcast
    't: BW-C-'        100 4 1 255 0, CHILD backweb
    DEFAULT NEXT S0"

sybaseWebsql  FLOW
               STATE-BASED

sybaseJdbc    FLOW
               STATE-BASED

sybaseTds     FLOW
               STATE-BASED

pointcast     FLOW
               STATE-BASED

backweb       FLOW
               STATE-BASED

mime          FLOW
               STATE-BASED
               STATES
               "S0: MATCH
                   'application' 900 0 0 1 0, CHILD
                   mimeApplication
                       'audio'     900 0 0 1 0, CHILD mimeTypeAudio
                       'image'     50 0 0 1 0, CHILD mimeTypeImage
                       'text'      50 0 0 1 0, CHILD mimeTypeText
                       'video'     50 0 0 1 0, CHILD mimeTypeVideo
                       'x-world'   500 4 1 255 0, CHILD
                   mimeTypeXworld
                   DEFAULT GOTO S0"

mimeApplication FLOW
                STATE-BASED

mimeAudio       FLOW
                STATE-BASED
                STATES
                "S0: MATCH
                    'basic'        100 0 0 1 0, CHILD
pdBasicAudio    'midi'          100 0 0 1 0, CHILD pdMidi
                'mpeg'          100 0 0 1 0, CHILD
pdMpeg2Audio    'vnd.rn-realaudio' 100 0 0 1 0, CHILD
pdRealAudio     'wav'           100 0 0 1 0, CHILD pdWav
                'x-aiff'        100 0 0 1 0, CHILD pdAiff

```

	'x-midi'	100 0 0 1 0, CHILD pdMidi
	'x-mpeg'	100 0 0 1 0, CHILD
pdMpeg2Audio		
	'x-mpgurl'	100 0 0 1 0, CHILD
pdMpeg3Audio		
	'x-pn-realaudio'	100 0 0 1 0, CHILD
pdRealAudio		
	'x-wav'	100 0 0 1 0, CHILD pdWav
	DEFAULT GOTO S0"	
mimeImage	FLOW	
	STATE-BASED	
mimeText	FLOW	
	STATE-BASED	
mimeVideo	FLOW	
	STATE-BASED	
mimeXworld	FLOW	
	STATE-BASED	
pdBasicAudio	FLOW	
	STATE-BASED	
pdMidi	FLOW	
	STATE-BASED	
pdMpeg2Audio	FLOW	
	STATE-BASED	
pdMpeg3Audio	FLOW	
	STATE-BASED	
pdRealAudio	FLOW	
	STATE-BASED	
pdWav	FLOW	
	STATE-BASED	
pdAiff	FLOW	
	STATE-BASED	

5. Supplemental Products

MeterWorks is supported by an optional Simple Network Management Protocol (SNMP) implementation. Envoy simplifies the process of porting *MeterWorks* to any platform. The *MeterWorks* SNMP Method Routines are written to directly interoperate with the Envoy product.

In addition, Envoy is supported by Emissary, Epilogue's optional MIB Compiler product. The MIB Compiler is a tool that greatly simplifies the creation and maintenance of proprietary MIB extensions. *MeterWorks* also takes direct advantage of the Emissary MIB compiler for making changes in the RMON groups that are supported.

Attaché, Epilogue's Portable UDP/IP protocol stack, complements *MeterWorks* and Envoy in environments that do not already provide a protocol stack for use by SNMP. Attaché version 3.0 provides full integration with *MeterWorks* version 4.00 and Envoy version 5.2, and fully implements MIB II (RFC 1213).

Index

Compiled Protocol Layout.....	5	Statement Types	See Definitions
Protocol Definition Language.....	5		

- **Exhibit B9:** State-based Sub-Classification Overview (document MFS-State-Classification.pdf) that describes the states of some of the protocols that are supported.

MeterFlow™ Traffic Classification System

State-based Sub-Classification Overview

Version A0.03



DRAFT - 03



NOTICE

This document contains confidential information proprietary to Technically Elite, Inc.

No part of its content may be used, copied, disclosed or conveyed to any party in any manner without prior written permission of Technically Elite, Inc.

Restricted Rights Legend

The programs and information contained herein are licensed only pursuant to a license agreement that contains use, reverse engineering, disclosure and other restrictions; accordingly, it is "Unpublished - all rights reserved under the applicable copyright laws."

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Licensed Programs clause at DFARS 52.227-7013.

Copyright © [REDACTED] by Technically Elite, Inc.

All Rights Reserved.

Printed in the United States of America.

Government Use

The Licensed Programs and their documentation were developed at private expense and no part of this is in the public domain.

The Licensed Programs are "Restricted Computer Software" as that term is defined in Clause 52.227-19 of the Federal Acquisition Regulations (FAR) and are "Commercial Computer Software" as that term is defined in Subpart 227.401 of the Department of Defense Federal Acquisition Regulation Supplement (DFARS).

- (i) If the licensed Programs are supplied to the Department of Defense (DoD), the Licensed Programs are classified as "Commercial computer Software" and the Government is acquiring only "restricted rights" in the Licensed Programs and their documentation as that term is defined in Clause 52.227-7013(c)(1) of the DFARS, and
- (ii) If the Licensed Programs are supplied to any unit or agency of the United States Government other than DoD, the Government's rights in the Licensed Programs and their documentation will be defined in Clause 52.227-19(c)(2) of the FAR.

All Technically Elite product names are trademarks or registered trademarks of Technically Elite, Inc. Other product names used in the manual are trademarks or registered trademarks of their respective holders.

Document revision A.03-004, format revision 1.00-000.

Technically Elite, Inc.

6330 San Ignacio Ave.

San Jose, CA 95119-1209, USA

Phone: +1.408.574.2300

Fax: +1.408.629.8300

E-mail: mworks@tecelite.com

URL: <http://www.tecelite.com>

Contents

1.	INTRODUCTION.....	5
2.	PROTOCOL CLASSIFICATION.....	7
2.1	CURRENT PROTOCOL CLASSIFICATIONS.....	7
2.2	IN-PROGRESS PROTOCOL CLASSIFICATIONS	7
3.	CURRENT PROTOCOL CLASSIFICATION	8
3.1	IP/IPIP/IPIP4 FRAGMENTATION	8
3.1.1	Features	8
3.1.2	Sub-classifications.....	8
3.1.3	Extensibility	8
3.1.4	Planned Developments	8
3.2	MICROSOFT ENDPOINT-MAPPER.....	9
3.2.1	Features	9
3.2.2	Sub-classifications.....	9
3.2.3	Extensibility	9
3.2.4	Planned Developments	10
3.3	SUNRPC PORTMAPPER	11
3.3.1	Features	11
3.3.2	Sub-classifications.....	11
3.3.3	Extensibility	11
3.3.4	Planned Developments	12
3.4	ORACLE 6/7 TRANSPARENT NETWORK SUBSTRATE (TNS).....	13
3.4.1	Features	13
3.4.2	Sub-classifications.....	14
3.4.3	Extensibility	14
3.4.4	Planned Developments	14
3.5	H.323 VIDEOCONFERENCING	15
3.5.1	Features	15
3.5.2	Sub-classifications.....	16
3.5.3	Extensibility	17
3.5.4	Planned Developments	17
3.6	HTTP	18
3.6.1	Features	18
3.6.2	Sub-classifications.....	19
3.6.3	Extensibility	21
3.6.4	Planned Developments	21
3.7	BACKWEB.....	22

3.7.1	<i>Features</i>	22
3.7.2	<i>Sub-classifications</i>	22
3.7.3	<i>Extensibility</i>	22
3.7.4	<i>Planned Developments</i>	22
4.	IN-PROGRESS PROTOCOL CLASSIFICATION	23
4.1	REAL-TIME STREAMING PROTOCOL (RTSP)	23
4.1.1	<i>Features</i>	23
4.1.2	<i>Sub-classifications</i>	24
4.1.3	<i>Extensibility</i>	25
4.2	NOVELL SERVICE ADVERTISING PROTOCOL (SAP)	26
4.2.1	<i>Features</i>	26
4.2.2	<i>Sub-classifications</i>	27
4.2.3	<i>Extensibility</i>	27
4.3	MS-MEDIA	28
4.3.1	<i>Features</i>	28
4.3.2	<i>Sub-classifications</i>	28
4.3.3	<i>Extensibility</i>	28
4.4	STREAMWORKS AND VDOLIVE	29
4.4.1	<i>Features</i>	29
4.4.2	<i>Sub-classifications</i>	29
4.4.3	<i>Extensibility</i>	29

1. Introduction

MeterFlow allows for a very rich set of protocol classification and sub-classification in the process of analyzing and interpreting Network Traffic. MeterFlow accomplishes this by combining the maintenance of state information with a robust ability to interpret network data streams.

Without the ability to maintain state, an increasingly large amount of Network Traffic will be misclassified, partially classified, or not classified at all with present traffic analysis and interpretation technologies. Pattern matching parser techniques employed in many such contemporary technologies provide little help here given the growing complexity of today's Network Traffic.

Misclassification is frequent given the practice of using assigned (or otherwise well-known) port/socket numbers as ephemeral ports/sockets. This has become especially noteworthy with the increasing proliferation of Web Browsers and MS WinSock. For example, BackWeb push-technology and Streamworks or VDOLive multimedia clients can use UDP ports that are either assigned to or used as defacto standards by other Network Applications such as Citrix, H.323 Gatekeeper, RealAudio, etc.

Partial classification is a common limitation in traffic analysis when the scope of interpretation is limited to a single packet. For example, one could see TCP Port #1527 referenced in a network packet and know that it was an Oracle TNS Packet. Without having interpreted the initial Oracle TNS protocol exchange spanning multiple packets, one could not have known that it was indeed PeopleSoft running over SQL*Net running over Oracle TNS.

Another example is of partial classification is simple "IP Fragmentation". Decoding the first fragment of an IP Datagram could easily determine that it further contained NFS over SunRPC over UDP. However, since subsequent fragments do not contain the UDP or SunRPC headers, they cannot be sub-classified for these protocols without having retained state and decoding information from the original (or first) fragment.

The inability to classify is becoming increasingly common as Network Applications use dynamic mechanisms to allocate and assign resources to various applications. There are a number of ways this can happen.

- In many cases, connections are established on a "truly" well-known port/socket of a server. The exchange on this connection serves to negotiate services requested/available and the address/port at which those services can be accessed. A second connection on the allocated/assigned address and port (almost always ephemeral) carries the bulk or volume of the data in the overall Network Session. Without the ability to interpret and analyze "data" in such allocation/assignment protocols connections, the volume traffic on the secondary connections cannot be distinguished from any other "un-interpretable" traffic. Microsoft's Endpoint-Mapper, SunRPC's Portmapper, and Oracle TNS are examples of such protocols.

- In other cases, available services and their locations (addresses and ports/sockets) are periodically announced. Without having interpreted and remembered the content of such announcements, traffic to/from them cannot be classified. Novell SAP and Apple's Name Binding Protocol (NBP) are examples of such announcement-based approaches.

The art of traffic classification becomes further complicated when a multitude of the underlying challenges described above occurs for the same Network Data events. For example, NFS version 1 is transferring one of its typical 32-Kbyte blocks of data in a single IP Datagram and is hence fragmenting it (partial classification scenario). This transfer is occurring on an "ephemeral" UDP port of the server that was allocated via an initial exchange with the SunRPC Portmapper protocol (no classification scenario). Or, even worse, the "ephemeral" UDP port on the server turns out to be the same as one of the defacto standard UDP ports that "RealAudio" uses (mis-classification scenario).

MeterFlow surmounts these challenges to provide accurate and thorough network traffic classification. This document discusses the currently supported and in-progress traffic classification capabilities of MeterFlow. It also presents the how MeterFlow may be extended to support further sub-classifications.

2. Protocol Classification

2.1 Current Protocol Classifications

MeterFlow currently includes support for classification and sub-classification of the following protocols, each of which are described in more detail in Section 3.0. In-progress developments to enhance or extend the sub-classification capabilities for these protocols are also described in Section 3.0.

1. IP Fragmentation
2. Microsoft Endpoint-Mapper
3. SunRPC Portmapper
4. Oracle 6/7 Transparent Network Substrate (TNS)
5. H.323 Video Conferencing
6. HTTP
7. BackWeb

2.2 In-Progress Protocol Classifications

Several new state-based protocol classification and sub-classification capabilities are under development. These protocols include the following and are described further in Section 4.0.

1. Real-Time Streaming Protocol (RTSP)
2. Novell Service Advertising Protocol (SAP)
3. MS Media
4. Streamworks and VDOLive

3. Current Protocol Classification

3.1 IP/IPIP/IPIP4 Fragmentation

3.1.1 Features

Fragmentation support for the Internet IP protocol is implemented in three MeterFlow sub-engines, each of which supports state maintenance and sub-classification retention for network packet fragments associated with the following protocols:

1. IP - Internet Protocol Version 4 datagram fragments
2. IPIP - IPIP datagram fragments Tunneled over IP
3. IPIP4 - IPIP4 datagram fragments Tunneled over IP

Key capabilities of these sub-engines include:

1. Tracking fragments for their corresponding protocols
2. Passing on 1st fragments through normal decoding and state-based decoding
3. Retaining complete 1st fragment sub-classification information for datagrams which are not further classified as state based (e.g. NFS Version 2 over UDP on well-known port 2049) and applying this information to all subsequent fragments components.
4. Retaining flow references for 1st fragment sub-classifications that further classify as state-based (e.g. Oracle TNS over TCP on a redirected, ephemeral port) and updating such flows for all subsequent fragment components.
5. Supporting concurrent fragmentation of data across multiple layers of Tunneling (e.g. IPIP4 fragments contained in IP fragments).

3.1.2 Sub-classifications

These sub-engines don't really "classify" or "sub-classify" underlying protocols contained in fragments beyond that normally done by the standard IP Version 4 decoding of the "protocol type". They do however retain "sub-classification" information or flow references as described above in Section 3.1.1.

3.1.3 Extensibility

By the nature of their scope, these sub-engines are not extensible beyond that which may be applied to standard IP Version 4 decoding with respect to the addition of new "Protocol Types".

3.1.4 Planned Developments

The "Internet Fragmentation Sub-Engines" will eventually add support for IP Version 6.

3.2 Microsoft Endpoint-Mapper

3.2.1 Features

The Microsoft Endpoint-Mapper **actually** supports the Endpoint-Mapper protocol defined by the “*Distributed Computing Environment (DCE) 1.1 – Remote Procedure Call*” specification. The key node point in the protocol directory for this protocol, and related applications determined by its mappings, is “*endpoint-mapper*”.

Key capabilities of this sub-engine include:

1. Tracking connections to and exchange within the well-known Endpoint-Mapper.
2. Distinguishing such “mapping” traffic from traffic on application connections subsequently “mapped”.
3. Detecting assignments of server application access assignments to various hosts and/or ports and creating sub-classifications for these access points.
4. When traffic to these access points is seen, it will be classified
 - a) By the appropriate application under “*endpoint-mapper*”, if the server application identifier in the mapping exchange is a **known** sub-application.
 - b) Minimally as “*endpoint-mapper*”, if the server application is unknown.
5. Allowing **known** sub-applications to be specified with respect to flow reporting with two levels of identification
 - a) Level 1 – Endpoint Mapped “Application Group”
 - b) Level 2 – Sub-application within the Application Group
6. Supporting the “connection-oriented” mode of Endpoint-Mapper operations.

3.2.2 Sub-classifications

Sub-classifications under “*endpoint-mapper*” include the following in both the “*tcp*” and “*udp*” protocol subtrees:

<i>endpoint-mapper</i>	→ <i>dcercpc-mapper</i>	(DCE RPC – Endpoint Mapping)
	→ <i>ms-exchange</i>	(MS-Exchange Directory)
	→ <i>information-store</i>	(MS-Exchange Information Store)
	→ <i>mta</i>	(MS-Exchange MS-Mail MTA)

3.2.3 Extensibility

New Sub-classifications are easily added as new entries in the DCE RPC Sub-Engine’s “Sub-Protocol Info” table, if the Universally Unique IDs (UUIDs) of the corresponding applications are known.

3.2.4 *Planned Developments*

Certainly there are more applications out there other than MS-EXCHANGE using DCE-RPC (also known as MS-RPC or Microsoft RPC since Microsoft adopted this RPC standard as opposed to SunRPC). As more notable applications are identified along with their assigned UUIDs, they will be added to the MeterFlow implementation.

Microsoft Exchange under the UDP instance of “endpoint-mapper” probably isn’t really a valid possibility. Accordingly, it will probably be removed from this instance.

Support for the “connection-less” mode of Endpoint Mapper operation could become a candidate for implementation when and if it can be determined that someone is indeed using it in the real world.

3.3 SunRPC Portmapper

3.3.1 Features

The SunRPC Portmapper protocol is defined by the “RPC: Remote Procedure Call Specification Version 2 (RFC 1831)” standard. The key node point in the protocol directory for this protocol, and related applications determined by its mappings, is “sunrpc”.

Key capabilities of this sub-engine include:

1. Tracking exchanges with the well-known SunRPC Portmapper.
2. Distinguishing such “mapping” traffic from traffic on application connections subsequently “mapped”.
3. Detecting assignments of server application access assignments to various hosts and/or ports and creating sub-classifications for these access points.
4. When traffic to these access points is seen, it will be classified
 - (a) By the appropriate application under “*sunrpc*”, if the server application identifier in the mapping exchange is a **known** sub-application.
 - (b) Minimally as “*sunrpc*”, if the server application is unknown.
5. Allowing known sub-applications to be specified with respect to flow reporting with a single levels of identification
 - (a) Level 1 – Portmapped “Application”

3.3.2 Sub-classifications

Sub-classifications under “sunrpc” include the following in both the “tcp” and “udp” protocol subtrees:

<i>sunrpc</i>	→ <i>portmapper</i>	(SunRPC – Port Mapping)
	→ <i>rstat</i>	(remote statistics)
	→ <i>nfs</i>	(network file service)
	→ <i>ypserv</i>	(yellow pages – server)
	→ <i>ypbind</i>	(yellow pages – bindings)
	→ <i>ypupdated</i>	(yellow pages – update daemon)
	→ <i>ypxferd</i>	(yellow pages – transfer daemon)
	→ <i>mount</i>	(remote file system mount)
	→ <i>3270-mapper</i>	(3270 terminal session mapper)
	→ <i>rje-mapper</i>	(remote job entry session mapper)
	→ <i>nis</i>	(next generation yellow pages)
	→ <i>pcnfsd</i>	(pcNFS daemon)

3.3.3 Extensibility

New Sub-classifications are easily added as new entries in the SunRPC Sub-Engine’s “Sub-Protocol Info” table, if the SunRPC Program Number of the corresponding applications are known.

3.3.4 *Planned Developments*

Certainly there are still other applications using SunRPC. As more notable applications are identified along with their assigned SunRPC Program Numbers, they will be added to the MeterFlow implementation.

Enhancement of the SunRPC Sub-Engine to additionally support SET, UNSET, DUMP, and/or CALLIT SunRPC Portmapper primitives could become a candidate for implementation when and if it can be determined that someone is indeed using them in the real world.

Improved understanding of whether the supported SunRPC sub-applications run over just UDP or TCP will enable the “sunrpc” sub-classifications to be more accurately categorized with respect to the protocol it actually runs over. For example, “portmapper”, “nfs”, and “pcnfsd” all operate only over UDP. Accordingly, their presence in the TCP subtree for “sunrpc” is unnecessary.

3.4 Oracle 6/7 Transparent Network Substrate (TNS)

3.4.1 Features

The Transparent Network Substrate (TNS) protocol is defined by Oracle Corporation and is used as the underlying networks access framework for its Oracle Version 6 and Oracle Version 7 database product offerings. The key node points in the protocol directory for this protocol and applications determined by its mappings are “oracl-tns”, “oracl-tns2”, “oracl-tns-srv”. These three node points reflect the three different “well-known” ports that serve to support initial access to Oracle TNS on Oracle Database servers. The first is a defacto, Oracle standard use. The next two access points (TCP ports) are assigned to Oracle by IANA.

Key capabilities of this sub-engine include:

1. Tracking connections to and exchanges in well-known Oracle TNS port traffic.
2. Learning the client application attempting to access the Oracle Database (e.g. PeopleSoft, Oracle Forms, etc.) to further classify traffic on the well-known Oracle TNS connections.
3. Detecting “redirections” of connections to various hosts and/or ports and creating sub-classifications for these access points. Such “redirections” inherit the sub-classifications of the initial connections to the well-known Oracle TNS service.
4. When traffic to these access points is seen or when TNS sessions are “accepted” on the well-know TNS service port, it will be classified
 - (a) By the appropriate client application under “*oracle-tns*” (or “*oracl-tns2*” or “*oracl-tns-srv*”), if the client application identifier is a **known** sub-application.
 - (b) Minimally as “*oracle-tns*” (or “*oracl-tns2*” or “*oracl-tns-srv*”), if the server application is unknown.
5. Allowing known sub-applications to be specified with respect to flow reporting with two levels of identification
 - (a) Level 1 – Oracle client’s “Application Group”
 - (b) Level 2 – Sub-application within the Application Group

3.4.2 Sub-classifications

Sub-classifications under “oracle-tns” include the following in the “tcp” subtree. Note that the same sub-classification occurs under the “oracl-tns2” and “oracl-tns-srv” nodes as well.

<i>oracle-tns</i>	→ <i>ms-odbc</i>	(<i>Microsoft ODBC</i>)
	→ <i>ms-ole</i>	(<i>Microsoft OLE</i>)
	→ <i>oracle-sqlplus</i>	(<i>Oracle SQLPlus</i>)
	→ <i>oracle-forms</i>	(<i>Oracle FORMS</i>)
	→ <i>peoplesoft</i>	(<i>PeopleSoft</i>)

3.4.3 Extensibility

New Sub-classifications are easily added as new entries in the Oracle TNS Sub-Engine’s “Sub-Protocol Info” table, if the Program Names (or names of the client programs’ executables) of the corresponding client applications are known.

3.4.4 Planned Developments

Further sub-classification of “PeopleSoft” is highly desired. Namely, breaking “peoplesoft” down into component applications.

Certainly there are still other native, client applications using Oracle TNS. As more notable applications are identified along with their assigned Program/Executable Names, they will be added to the MeterFlow implementation.

“SAP R/3” and “Baan”, in particular, are high priority applications to establish such additional support for.

A major enhancement to the Oracle TNS sub-engine will be to further build upon the application sub-classification capabilities presently supported. This will allow the sub-engine to further delve into the SQL*Net content to determine the actual client applications riding atop 4GL tools (such as Oracle FORMs) and access APIs (such as MS ODBC, and MS OLE).

The three Oracle TNS subtrees (“oracle-tns”, “oracl-tns2”, and “oracl-tns-srv”) will most likely be consolidated under a single subtree under TCP (“oracle-tns”).

3.5 H.323 Videoconferencing

3.5.1 Features

H.323 is an umbrella standard published by the International Telecommunication Union (ITU, formerly CCITT) for videoconferencing. H.323 entails one of the most complicated traffic classification challenges of today's networking protocols. This arises from its inherent multi-tier connection/data-stream architecture.

In H.323, connections are initially established on a well-known service port. The Q.931 protocol is used on this "H.323 Call Setup" connection to set-up a second connection on an ephemeral port. The second "H.323 Call Control" connection uses the H.245 protocol to negotiate audio and video capabilities (codecs) as well as to further set-up RTP/RTCP audio and video data streams over ephemeral UDP ports.

Key capabilities of this sub-engine include:

1. Tracking connections to and exchanges on well-known H.323-host-call port (Q.931 protocol) traffic.
2. Detecting assignments of H.245 access points to various hosts and/or ports and creating H.245 sub-classifications for these access points.
3. Tracking connections to and exchanges with such assigned H.245 access points.
4. Detecting the assignment of RTP/RTCP audio and video, UDP datastreams access points as well as the audio and video "codecs" negotiated for use on them and creating RTP/RTCP sub-classifications for these access points.
5. When traffic to these RTP/RTCP access points are seen it will be classified
 - (a) By the appropriate "codec" under "*rtp*", if the negotiated codec is a **known** audio/video stream type.
 - (b) Minimally as "*rtp*", if the negotiated codec is unknown.
6. Allowing known sub-applications (audio/video datastreams) to be specified with respect to flow reporting with three levels of identification
 - (a) Level 1 – Datastream Class (e.g. audio, video, other...)
 - (b) Level 2 – Datastream Type within the Datastream Class
 - (c) Level 3 – Datastream Sub-Type within the Datastream Type
7. Supporting the Q.931 "normal mode" of operation for "H.323 Call Setup connections".

3.5.2 Sub-classifications

“H.323 Call Setup” Sub-classifications under “h323-host-call” include the following in the “tcp” subtree.

<i>h323-host-call</i>	→ <i>q931</i>	(H.323 Call Setup)
	→ <i>q931-fast-start</i>	(H.323 Combined Setup and Control)

“H.323 Call Control” Sub-classifications under “h323-host-control” include the following in the “tcp” subtree.

<i>h323-host-control</i>	→ <i>h245</i>	(H.323 Call Control)
--------------------------	---------------	----------------------

Audio and video datastream Sub-classifications under “RTP/RTCP” include the following in the “udp” subtree.

<i>RTCP</i>	→		(Audio/Video Stream Control sub-channel)
<i>RTP</i>	→ <i>audio</i>	→ <i>G.711</i>	(Audio Transfer sub-channel)
		→ <i>G.722</i>	
		→ <i>G.728</i>	
		→ <i>G.729</i>	
		→ <i>MPEG1-audio</i>	
		→ <i>G.723</i>	
		→ <i>GSM</i>	
	→ <i>video</i>	→ <i>H.261</i>	→ <i>QCIF</i> (Video Transfer sub-channel)
			→ <i>CIF</i>
		→ <i>H.263</i>	→ <i>SQCIF</i>
			→ <i>QCIF</i>
			→ <i>CIF</i>
			→ <i>4CIF</i>
			→ <i>16CIF</i>
		→ <i>MRV</i>	

Standards for the audio stream sub-classifications indicated above are:

G.711 - 64 Kbps, 8K samples/sec, 8-bit companded PCM (A-law or μ -law), high quality, low complexity. Required for H.320 and H.323.

G.722 - ADPCM audio encode/decode (64 kbit/s, 7 kHz) .

G.723 - Speech coder at 6.3 and 5.3 Kbps data rate. Medium complexity. Required for H.324; Optional for H.323.

G.728 - 16 Kbps, LD-CELP, high quality speech coder, very high complexity. Optional for H.320 and H.323.

G.729 - 8Kbps, LD-CELP, high quality speech coder, medium complexity. G.DSVD is an interoperable subset.

GSM - Group Special Mobile -- European telephony standard, not ITU. Used by ProShare Video Conferencing software versions 1.0-1.8. 13Kbps, medium quality for voice only, low complexity.

Standards for the **video** stream sub-classifications indicated above are:

- H.261 - Supports 352x288 (CIF or FCIF) and 176x144 (QCIF). DCT-based algorithm tuned for 2B to 6B ISDN communication. Required for H.320, H.323, and H.324.
- H.263 - Much-improved derivative of H.261, tuned for POTS data rates. Mostly aimed at QCIF and Sub-QCIF (128x96 -- SQCIF). Optional for H.323 and H.324, although industry is focusing on it for POTS. Being added as an option to H.261.
- MRV - Intel Indeo® video compression technology tuned for ISDN and LAN data rates.

3.5.3 Extensibility

New Sub-classifications are easily added as new entries in the H.323 Sub-Engine's "Sub-Protocol Info" table, if the Audio/Video Capability Identifiers of the corresponding audio/video datastream are known.

3.5.4 Planned Developments

There are still more audio/video datastream formats. As others are identified along with their assigned capability identifiers, they will be added to the MeterFlow implementation.

There is a mode of H.323 operation defined called "Q.931 Fast Start". In this mode, "H.323 Call Control" operations (normally performed under their own H.245 connection) are piggybacked over Q.931 in the "H.323 Call Setup" connection. The use of this mode of operation has historically been rare and infrequent in contemporary videoconferencing products. The H.323 sub-engine will be enhanced to support this mode of operation.

3.6 HTTP

3.6.1 Features

The HTTP Protocol is the basis of common, present-day Web Browsers and has become a fundamental transport mechanism for many Internet applications. HTTP operates over TCP connections. Traditional/typical use of HTTP involves the establishment/tear-down of an individual HTTP connection for each element of exchange in a given user session activity (e.g. a web page will involve many TCP connections to effect the transfer of the various components of the activity).

There are two ways to distinguish the nature of the application information involved in an HTTP exchange.

1. HTTP content type
2. Interpretation of various fields in the HTTP data

Key capabilities of this sub-engine include:

1. Tracking connections to and exchanges in well-known HTTP Port traffic.
2. Learning the nature of the application data being transferred or accessed to further classify traffic on such well-known HTTP connections.
3. Learning the nature of the application by virtue of analyzing selected HTTP fields.
4. Allowing known sub-applications to be specified with respect to flow reporting with two levels of identification
 - (a) Level 1 – HTTP sub-application group (e.g. database, application, video, etc...)
 - (b) Level 2 – sub-application within the sub-application group
5. Classifying HTTP traffic
 - (a) By the appropriate sub-application within the sub-application group, if the sub-application identifier is **known**.
 - (b) Minimally by the sub-application group, if the negotiated sub-application identifier is unknown.

3.6.2 Sub-classifications

Sub-classifications under “*www-http*” include the following in the “*tcp*” subtree. Note that the same sub-classification occurs under the “*alternate-http*” node as well.

```

www-http → database    → sybase-web-sql
                        → sybase-tunneled-ids
                        → jdbc          → odbc-bridge
                                           → ibm-db2
                                           → gupta-jdbc
                                           → sybase-jdbc
→ application → pointcast
               → backweb
               → datawindow
               → edi-content
               → edi-x12
               → edifact
               → excel
               → macbinhex40
               → mp3
               → mspowerpoint
               → msword
               → news-message-id
               → news-transmission
               → octet-stream
               → oda
               → pdf
               → postscript
               → powerbuilder
               → quattro-pro
               → rtf
               → sgml
               → vnd-framemaker
               → vnd-lotus-1-2-3
               → vnd-lotus-approach
               → vnd-lotus-freelance
               → vnd-lotus-organizer
               → vnd-lotus-wordpro
               → vnd-mif
               → vnd-ms-excel
               → vnd-ms-powerpoint
               → vnd-ms-project
               → vnd-ms-word
               → vnd-verbuilder
               → vnd-rn-realplyer
               → vnd-visio
               → wordperfect
               → x-bcpio
               → x-compress
               → x-cpio
               → x-csh
               → x-director
               → x-dvi
               → x-gtar
               → x-gzip
               → x-javascript
               → x-latex
               → x-lotus-notes
               → x-macbinary
               → x-mif
               → x-pnclmd
               → x-pn-realaudio
               → x-powerpoint
               → x-sh
               → x-stuffit
               → x-tar
               → x-tcl

```

<i>www-http</i>	→ <i>application</i>	→ <i>x-tex</i>
(<i>cont.</i>)	(<i>cont.</i>)	→ <i>x-troff</i>
		→ <i>x-ustar</i>
		→ <i>x-zip-compressed</i>
		→ <i>xpp5</i>
		→ <i>zip-archive</i>
		→ <i>x-netcdf</i>
	→ <i>audio</i>	→ <i>basic</i>
		→ <i>midi</i>
		→ <i>mpeg</i>
		→ <i>vnd-rn-realaudio</i>
		→ <i>wav</i>
		→ <i>x-aiff</i>
		→ <i>x-midi</i>
		→ <i>x-mpeg</i>
		→ <i>x-mpgurl</i>
		→ <i>x-pn-realaudio</i>
		→ <i>x-wav</i>
	→ <i>image</i>	→ <i>cgm</i>
		→ <i>g3fax</i>
		→ <i>gif</i>
		→ <i>ief</i>
		→ <i>jpeg</i>
		→ <i>pict</i>
		→ <i>png</i>
		→ <i>tiff</i>
		→ <i>vnd-rn-realflash</i>
		→ <i>vnd-rn-realpix</i>
		→ <i>x-bitmap</i>
		→ <i>x-pixmap</i>
		→ <i>x-quicktime</i>
		→ <i>x-windowdump</i>
		→ <i>x-xbm</i>
	→ <i>text</i>	→ <i>enriched</i>
		→ <i>html</i>
		→ <i>plain</i>
		→ <i>richtext</i>
		→ <i>sgml</i>
		→ <i>tab-separated-value</i>
		→ <i>vnd-rn-text</i>
		→ <i>css</i>
	→ <i>video</i>	→ <i>avi</i>
		→ <i>mpeg</i>
		→ <i>msvideo</i>
		→ <i>ms-video</i>
		→ <i>quicktime</i>
		→ <i>vnd-rn-realvideo</i>
		→ <i>vnd-vivo</i>
		→ <i>x-ls-asf</i>
		→ <i>x-ls-asx</i>
		→ <i>x-mpeg</i>
		→ <i>x-ms-asf</i>
		→ <i>x-ms-asx</i>
		→ <i>x-msvideo</i>
		→ <i>x-sgi-movie</i>
	→ <i>x-world</i>	→ <i>x-vrml</i>

3.6.3 Extensibility

New Sub-classifications require much more thought and analysis when being added to HTTP. This arises from the following factors:

1. HTTP is a “text” based protocol
2. To support “minimum” execution overhead, when searching the HTTP Sub-Engine's “Sub-Protocol Info” database, a rather robust set of sequentially indexed, look-aside tables are employed.
 - (a) The challenge here is to take a string from an HTTP packet (e.g. Content Type) and match it with any one of approximately 110+ well known (as is the case with Content Type)
 - (b) And to do so within an embedded environment that is trying to keep up with the network packet rate at line speed.
 - (c) The supported search mechanism can identify a single match candidate sub-string by looking at typically no more than 3 to 5 characters of the sub-string from the HTTP packet.
3. Adding a sub-classification to the HTTP “Sub-Protocol Info” Database is simply a matter of adding a new entry if the “Content Type” or “JDBC URL Component” is known.
4. Updating and/or extending the “look-aside” tables requires extreme caution and accuracy.

Extensibility for this sub-engine will be **tremendously improved** when the PDL compiler is incorporated for this sub-engine.

3.6.4 Planned Developments

New “Content Types” are springing up almost every week. As new applications are identified along with their designated Content Types, they will be added to the MeterFlow implementation.

WebNFS from Sun Microsystems, Inc. tunnels NFS file access over HTTP and is a clear candidate for inclusion into this sub-engine.

There are many other JDBC packages from various database manufactures and technology suppliers that are integrated with WWW. Oracle's being the most noted at this time. As more are identified along with their designated JDBC URL Selectors, they will be added to the MeterFlow implementation.

3.7 BackWeb

3.7.1 Features

BackWeb (BackWeb Technologies, Inc.) is a news/broadcast application. It may be configured to operate in either of 2 modes:

- a) HTTP only (see Section 3.6 above)
- b) UDP for access to BackWeb Servers & HTTP to access to 3rd party channels (polite mode)

BackWeb operates over UDP in what it calls its “Polite Client” mode. In this mode, BackWeb has an unusual mechanism of exchange that makes traffic in one direction very easy to see (well-known), but difficult to classify in the other direction.

The BackWeb sub-engine has been implemented specifically for BackWeb’s UDP (Polite Mode) access protocol.

Key capabilities of this sub-engine include:

1. Tracking exchanges with BackWeb Servers in well-known BackWeb Server port traffic.
2. Remembering the access points of traffic from BackWeb Clients and creating sub-classifications for these access points.
3. When traffic to these access points are seen, it will be classified
 - (a) as “backweb”

3.7.2 Sub-classifications

BackWeb (Polite Mode) traffic is classified as “backweb” in the “udp” subtree. No further sub-classifications for BackWeb are supported.

3.7.3 Extensibility

There are no known Sub-Classifications to be supported for BackWeb at this time.

3.7.4 Planned Developments

No further development efforts are currently planned for the BackWeb sub-engine.

4. In-Progress Protocol Classification

4.1 Real-Time Streaming Protocol (RTSP)

4.1.1 Features

The “Real-Time Streaming Protocol” is defined in RFC 2326. Like HTTP it is a “text” based protocol. Unlike HTTP, its principle purpose is to enable the controlled, on-demand delivery of real-time data, such as audio and video.

In function it acts similar to H.323’s “Call Setup” and “Call Control” services, however, in a single connection on a well-known port. Ultimately, it serves to set up RTP/RTCP datastreams over UDP.

Key capabilities of this sub-engine include:

1. Tracking exchanges with the well-known RTSP server.
2. Detecting the assignment of RTP/RTCP audio and video, UDP datastreams access points as well as the audio and video “codecs” negotiated for use on them and creating RTP/RTCP sub-classifications for these access points.
3. When traffic to these RTP/RTCP access points are seen it will be classified
 - (a) By the appropriate “codec” under “*rtp*”, if the negotiated codec is a **known** audio/video stream type.
 - (b) Minimally as “*rtp*”, if the negotiated codec is unknown.
4. Allowing known sub-applications (audio/video datastreams) to be specified with respect to flow reporting with three levels of identification
 - (a) Level 1 – Datastream Class (e.g. audio, video, other...)
 - (b) Level 2 – Datastream Type within the Datastream Class
 - (a) Level 3 – Datastream Sub-Type within the Datastream Type

4.1.2 Sub-classifications

RTSP traffic will be classified as "rtsp" in the "tcp" subtree. No further sub-classifications for RTSP are supported.

NEW Audio and video datastream Sub-classifications under "RTP" will include the following in the "udp" subtree.

<p>RTP → audio</p>	<p>→ 1016 → DVI4 → L8 → L16 → LPC → MPA → VDMI → AIFF-C</p>	<p>(Audio Transfer sub-channel)</p>
<p>→ video</p>	<p>→ CelB → JPEG → MPV → MP2T → nv</p>	<p>(Video Transfer sub-channel)</p>

Standards for the audio stream sub-classifications indicated above are:

- 1016 - frame based encoding using code-excited linear prediction (CELP) and is specified in Federal Standard FED-STD 1016
- DVI4 - IMA ADPCM wave type, "IMA Recommended Practices for Enhancing Digital Audio Compatibility in Multimedia Systems (version 3.0)"
- L8 - L8 denotes linear audio data, using 8-bits of precision with an offset of 128, that is, the most negative signal is encoded as zero.
- L16 - L16 denotes uncompressed audio data, using 16-bit signed representation with 65535 equally divided steps between minimum and maximum signal level, ranging from -32768 to 32767. The value is represented in two's complement notation and network byte order.
- LPC - LPC designates an experimental linear predictive encoding contributed by Ron Frederick, Xerox PARC, which is based on an implementation written by Ron Zuckerman, Motorola, posted to the Usenet group comp.dsp on June 26, 1992.
- MPA - MPA denotes MPEG-I or MPEG-II audio encapsulated as elementary streams. The encoding is defined in ISO standards ISO/IEC 11172-3 and 13818-3. The encapsulation is specified in work in progress.
- VDVI - VDVI is a variable-rate version of DVI4, yielding speech bit rates of between 10 and 25 kb/s. It is specified for single-channel operation only.

AIFF-c - Apple Computer, "Audio interchange file format AIFF-C," Aug. 1991. (also <ftp://ftp.sgi.com/sgi/aiff-c.9.26.91.ps.Z>).

Standards for the **video** stream sub-classifications indicated above are:

CelB - The CELL-B encoding is a proprietary encoding proposed by Sun Microsystems. "RTP payload format of CellB video encoding," Work in Progress, Internet Engineering Task Force, Aug. 1995.

JPEG - The encoding is specified in ISO Standards 10918-1 and 10918-2.

MPV - Designates the use MPEG-I and MPEG-II video encoding elementary streams as specified in ISO Standards ISO/IEC 11172 and 13818-2, respectively.

MP2T - MP2T designates the use of MPEG-II transport streams, for either audio or video.

nv - The encoding is implemented in the program 'nv', version 4, developed at Xerox PARC

4.1.3 Extensibility

New Sub-classifications will easily added as new entries in the RTSP Sub-Engine's "Sub-Protocol Info" table, if the Payload Types of the corresponding audio/video stream are known.

4.2 Novell Service Advertising Protocol (SAP)

4.2.1 Features

The Novell Service Advertising Protocol (SAP) is a protocol similar in nature to the “SUN RPC Portmapper” protocol. It is used to support the dynamic management and locating of “services” with regards to their locations (network addresses) and port assignments.

SAP uses a completely different protocol than the SUN RPC protocol Portmapper. Also, a fundamental difference from Sun RPC is that SAP periodically broadcasts services that are in it’s advertising database.

Key capabilities of this sub-engine include:

1. Tracking SAP announcements periodically broadcast by Novell Netware servers.
2. Distinguishing such “announcement” traffic from traffic on application connections subsequently “mapped”.
3. Detecting assignments of server application access assignments to various hosts and/or sockets and creating sub-classifications for these access points.
4. When traffic to these access points is seen, it will be classified
 - (a) By the appropriate application under “*nov-sap*”, if the server application identifier in the announcement is a **known** sub-application.
 - (b) Minimally as “*nov-sap*”, if the server application is unknown.
5. Allowing known sub-applications to be specified with respect to flow reporting with a single levels of identification
 - (a) Level 1 – SAP Mapped “Application Group”
 - (b) Level 2 – Sub-application within the Application Group

4.2.2 Sub-classifications

Sub-classifications under “nov-sap” will include the following in the “ipx.nov-pep” subtree.

<i>nov-sap</i>	→ <i>announce</i>	(Novell SAP Announcements)
	→ <i>ms-exchange</i>	(Microsoft Exchange)
	→ <i>sybase_sqlany</i>	(Sybase SQL Anywhere)
	→ <i>sybase_sqlenterprise</i>	(Sybase SQL Enterprise)
	→ <i>gupta-sqlbase</i>	(Gupta SQLBase)
	→ <i>ms-sna-server</i>	(Microsoft SNA Server)
	→ <i>ms-sql-server</i>	(Microsoft SQL Server)
	→ <i>citrix-app-server</i>	(Citrix Application Server)
	→ <i>citrix-app-server-nt</i>	(Citrix Application Server for NT)
	→ <i>hp-laserjet</i>	(HP Laserjet Printer)
	→ <i>advertising-print-svr</i>	(Advertising Print Server)
	→ <i>netware-sql-server</i>	(Novell Netware SQL Server)
	→ <i>remote-bridge</i>	(Remote Bridge Router Service)
	→ <i>bridge-server</i>	(Bridge Server)
	→ <i>print-queue</i>	(Print Queue Server)

4.2.3 Extensibility

New Sub-classifications will easily added as new entries in the Novell SAP Sub-Engine’s “Sub-Protocol Info” table, if the SAP IDs of the corresponding application are known.

4.3 MS-Media

4.3.1 Features

MS-Media is a audio/video streaming, multimedia application (similar to RealAudio) from Microsoft. MS-Media may be configured to operate over UDP when transferring its payload. In this configuration, MS-Media has an unusual mechanism to allocate UDP resources for this purpose via an initial TCP connection.

The MS-Media sub-engine will be implemented specifically for MS-Media's access protocol.

Key capabilities of this sub-engine include:

1. Tracking connections to and exchanges in well-known MS-Media port traffic.
2. Detecting assignments of UDP access points to various hosts and/or ports and creating MS-Media sub-classifications for these access points.
3. When traffic to these access points are seen, it will be classified
 - (a) as "ms-media"

4.3.2 Sub-classifications

Such MS-Media traffic will be classified as "*ms-media*" in the "*udp*" subtree. No further sub-classifications for MS-Media will be initially supported.

4.3.3 Extensibility

Sub-Classification is beyond the initial scope of the MS-Media sub-engine. Eventually, the sub-engine will be able to sub-classify the types of payloads being transferred.

4.4 Streamworks and VDOLive

4.4.1 Features

Streamworks and VDOLive are multi-media, streaming applications which transfer their payloads over UDP.

Like BackWeb, Streamworks and VDOLive employ unusual mechanisms of exchange that makes traffic one direction very easy to see (well-known), but difficult to classify in the other direction.

The BackWeb sub-engine will be expanded to further support Streamworks and VDOLive classification.

For a description of the key capabilities of the sub-engine, see Section 3.7:

4.4.2 Sub-classifications

Streamworks and VDOLive traffic is classified as “streamworks-xing-mpeg” and “vdolive” in the “udp” subtree; respectively. No further sub-classifications for these protocols will be supported.

4.4.3 Extensibility

Sub-Classification is beyond the initial scope Streamworks and VDOLive. Eventually, the sub-engine will be able to sub-classify the types of payloads being transferred.